X6-RX User's Manual

X6-RX User's Manual

The X6-RX User's Manual was prepared by the technical staff of
Innovative Integration on April 16, 2011.

For further assistance contact:

Innovative Integration
2390-A Ward Ave
Simi Valley, California 93065

PH: (805) 578-4260
FAX: (805) 578-4225

email: techsprt@innovative-dsp.com
Website: www.innovative-dsp.com

This document is copyright 2011 by Innovative Integration. All rights
are reserved.

VSS \ Distributions \ X6-RX \ Documentation \ Manual \ X6-RXMaster.odm

#XXXXXX

Revision	Date	Author	Notes
1.0	11/19/10	DPM	Initial release.
1.1	04/16/11	DPM	Updated hardware discussion for current revision. Updated power discussion. Cleanup and formatting.

Table of Contents

List of Tables....8
List of Figures....10
Chapter 1:Introduction....12
Real Time Solutions!....12
Vocabulary....12
What is X6-RX? 13
What is Malibu? 13
What is C++ Builder?....14
What is DialogBlocks?....14
What is wxWidgets?....14
What is Qt?....14
What is Microsoft MSVC?....15
What kinds of applications are possible with Innovative Integration hardware?....15
Why do I need to use Malibu with my Baseboard?....15
Finding detailed information on Malibu....15
Online Help....16
Innovative Integration Technical Support....16
Innovative Integration Web Site....16
Typographic Conventions....16
Chapter 2:Windows Installation....18
Host Hardware Requirements....18
Software Installation....18
Starting the Installation 19
The Installer Program....20
Tools Registration....22
Bus Master Memory Reservation Applet....22
Hardware Installation....23
After Power-up....24
Installation on a Deployed System....24
Running MalibuRed....24
Chapter 1:Installation on Linux....26
Package File Names....26
Prerequisites for Installation....26
The Redistribution Package Group - MalibuRed....26
Malibu....27
Other Software....27
Baseboard Package Installation Procedure....27
Board Packages....28
Unpacking the Package....28
Creating Symbolic Links....28
Completing the Board Install....29
Linux Directory Structure....29
Applets....29
Documentation....29

Examples....29
Hardware....29
Chapter 2:Hardware Installation....30
Compatible Host Cards....30
System Requirements....33
Power Considerations....34
Mechanical Considerations....35
Chapter 3:About the X6 XMC Modules....36
X6 XMC Architecture....36
X6 Computing Core....38
X6 PCI/PCI Express Interface....39
PCI Express Interface....40
PCI Interface....40
Configuring The Host Interface....41
PCI and PCI Express Standards....41
FPGA Integration with the PCI/PCIe Host Data Path....42
Data Buffering and Memory Use....42
DRAM Devices....43
DRAM Connections to FPGA....43
Data Buffer DRAM....43
Serial FLASH Interface....44
FLASH Memory Description....44
Digital I/O....45
Digital IO Connections....46
Digital IO Electrical Characteristics....46
LVCMOS 2.5V Pins....46
LVDS 2.5 Pins....47
Software Support....47
Notes on Digital IO Use....48
P16 Communication Ports....48
GTX Connections to P16....49
GTX Electrical Characteristics....49
GTX Reference Clocks....50
Aurora Ports....51
Connecting Multiple X6 Modules Using Aurora Ports....52
System Timing Features on P16....53
Thermal Protection and Monitoring....54
Temperature Readout....56
Software Support for Temperature Monitoring....56
System Thermal Design....57
Fan Control....57
Conduction Cooling....58
X6 FPGA Heat Spreader....60
X6 Power Saving Features....60
Power Supply Controls....61
Led Indicators....62

LEDs NOT Lit with FrameWork Logic Installed....62
JTAG Scan Path....62
FrameWork Logic....63
Integrating with Host Cards and Systems....63
Standalone Operation....64
Logic Configuration....65
Logic Configuration from FLASH EEPROM....65
Logic Image Selection....66
Updating the Logic Configuration FLASH EEPROM....67
Rescuing the Card When the Logic Image is Bad....68
Chapter 4:Writing Custom Applications....69
The Snap Example....69
Tools Required....69
Program Design....70
The Host Application 70
User Interface....70
Configure Tab....70
Setup Tab....71
Stream Tab....73
Host Side Program Organization....74
ApplicationIo....75
Initialization....75
Starting Data flow....79
Handle Data Available....82
The Wave Example....83
User Interface....83
The Setup Tab....83
The Waveform Tab....84
Stream Tab....85
ApplicationIo....85
Stream Preconfigure....85
Start Streaming....86
Data Required Event Handler....89
FillWaveformBuffer()....90
Developing Host Applications....93
Borland Turbo C++....93
Other considerations:....94
Microsoft Visual Studio 2005....95
DialogBlocks....97
Summary....97
Chapter 5:Vita Packet Format....98
Overview....98
X6 Velocia Packets....98
Packet Header Format....98
Packet Data Format....99
X6 Vita Packets....99

Packet Header Format **100**
VITA Header IF word 100
VITA Header SID word 101
VITA Header Class OUI Word 102
Vita Header Class Info Word 102
Vita Header Timestamp – Integer Seconds Word 102
Vita Header Timestamp – Fractional Seconds High and Low Words 102
Vita Packet Trailer Format **102**
VITA Trailer Word 102
State and Event Bits and Enable Bits 102
Bits 20-23 103
Context Packet Count 103
Padding 103
Chapter 6:Applets **104**
Common Applets **105**
Registration Utility (NewUser.exe) **105**
Reserve Memory Applet (ReserveMemDsp.exe) **106**
Data Analysis Applets **106**
Binary File Viewer Utility (BinView.exe) **106**
Chapter 7:Applets for the X6 Family Baseboards **107**
Logic Update Utility (VsProm.exe) **107**
Finder **108**
Chapter 8:X6-RX XMC Module **109**
Introduction **109**
Hardware Features **111**
A/D Converters **111**
A/D Front End **111**
Input Range and Conversion Codes **112**
Driving the A/D Inputs **112**
Overrange Detection **113**
Sample Rate Generation and Clocking Controls **113**
External Clock/Reference Input **114**
Sample Rate Generation **114**
Setting the Sample Rate in the SNAP Example **114**
Controlling the PLL **115**
How To Set the Sample Rate Generator to a Specific Frequency **115**
Using An External PLL Reference **116**
Using An External Clock for Sample Clock **118**
External Clock Requirements **119**
CDCDE72010 SPI Port **120**
VCXO I2C Port **120**
Triggering **121**
Trigger Sources **122**
Front Panel External Trigger Input Requirements **122**
Framed Trigger Mode **123**
Decimation **123**

Trigger Controls in SNAP Example....123
Multi-A/D Synchronization....124
Digital Downconverter (DDC) ASIC....125
FrameWork Logic Functionality....125
Alert Log....127
Overview....127
Types of Alerts....127
Alert Packet Format....128
Software Support for Alerts....129
Tagging the Data Stream....129
Calibration....129
Updating the Calibration Coefficients....129
Using the X6-RX....130
Where to start?....130
Getting Good Analog Performance....130
Power Requirements....132
Power Supplies....132
Power Consumption....132
Environmental Ratings....134
Performance Data....136
Analog Input Performance Summary....136
Connectors....139
Front Panel Connectors J1-J6....139
XMC P15 Connector....141
XMC P16 Connector....144
PMC JN1 Connector....147
PMC JN2 Connector....150
PMC JN4 Connector....153
Xilinx JTAG Connector....155
FLASH Boot Image Select....156
Mechanicals....156

List of Tables

Table 1. X6 XMC Bus Requirements....30
Table 2. Required PCIe Resource Allocations....34
Table 3. XMC Mounting Hardware....35
Table 4. X6 XMC Family....37
Table 5. X6 XMC Family....37
Table 6. X6 XMC Family Peripherals....38
Table 7. X6 Computing Core Devices....38
Table 8. Logic Images for PCI and PCI Express Hosts....41
Table 9. PCI/PCI Express Standards Compliance....42
Table 10. Interfaces from PCI Express to Application Logic....42
Table 11. X6 DRAM Devices....43
Table 12. Buffer Memory Bandwidth....44
Table 13. X6 Serial FLASH Device....44
Table 14. DIO Connector Configurations....46
Table 15. LVCMOS 2.5 Digital IO Bits Electrical Characteristics....46
Table 16. LVDS2.5 Digital IO Bits Electrical Characteristics....47
Table 17. IUsesDioPort Class Operations....47
Table 18. P16 GTX Assignments and Use....49
Table 19. GTX Signal Electrical Characteristics....50
Table 20. GTX PLL Reference Clock Characteristics....50
Table 21. GTX Alternate Reference Clock Input Characteristics....51
Table 22. System Integration Signals on P16 Connector....54
Table 23. X6 Temperature Threshold and Limits 55
Table 24. X6 Temperature Sensor....56
Table 25. X6 Conduction Cooling Hardware....60
Table 26. X6 Power Control Features....61
Table 27. X6 Power Supply Controls....62
Table 28. X6 LED Indicators....62
Table 29. X6 JTAG Scan Path....63
Table 30. XMC Adapters and Hosts....64
Table 31. X6 FPGA Logic Image Sizes and Configuration Times....66
Table 32. Logic Image Selection Jumper JP1....67
Table 33. Development Tools for the Windows Snap Example....69
Table 34. X6 Velocia (Velo) Packet Header....98
Table 35. Velocia Header Word....99
Table 36. Vita Packet Format....99
Table 37. Timestamp Integer Seconds Options 101
Table 38. Timestamp Fractional Seconds Options 101
Table 39. Padding Example 103
Table 40. Maximum Padding for X6 Boards....103
Table 41. X6-RX A/D Features....111
Table 42. High Speed A/D Conversion Coding....112
Table 43. PLL Specifications....114
Table 44. Sample Rate Generation Parameters....116

Table 45. Steps to Configure the VCXO and PLL 116
Table 46. Effect of Sample Clock Jitter on Digitizing Accuracy (Courtesy Analog Devices, Inc.)....117
Table 47. External PLL Reference Additive Jitter and Delay....117
Table 48. External PLL Reference Requirements....118
Table 49. External Clock Input Requirements....120
Table 50. PLL SPI interface – 0x801(r/w)....120
Table 51. VCXO control and I2C register – 0x802 (r/w)....121
Table 52. Trigger Modes....121
Table 53. X6-RX Trigger Sources....122
Table 54. External Trigger Input Requirements....123
Table 55. Alert Types....127
Table 56. Alert Packet Format....128
Table 57. X6-RX Power Supply Requirements....132
Table 58. X6-RX Baseline Power Consumption....132
Table 59. X6-RX Power Consumption for IO Features....133
Table 60. X6-RX Environmental Limits....134
Table 61. X6-RX Analog Performance Summary....136
Table 62. X6-RX XMC Connector P15 Pinout....142
Table 63. P15 Signal Descriptions....143
Table 64. X6-RX XMC Secondary Connector P16 Pinout....145
Table 65. P16 Signal Descriptions....146
Table 66. X6 JN1 Connector Pinout....148
Table 67. X6 JN2 Connector Pinout....151
Table 68. X6 JN2 Connector Pinout....154
Table 69. X6-RX JP3 Xilinx JTAG Connector Pinout....156
Table 70. X6-RX JP1 Boot Image Select....156

List of Figures

Figure 1. Vista Verification Dialog....19
Figure 2. Innovative Install Program....20
Figure 3. Progress is shown for each section....21
Figure 4. ToolSet registration form....22
Figure 5. BusMaster configuration....23
Figure 6. Installation complete....23
Figure 7. Innovative x8 lane PCIe – XMC.3 adapter card (P/N 80259)....31
Figure 8. Innovative 3U VPX- XMC.3 adapter card (P/N 80260) 31
Figure 9. Innovative Single lane PCIe – XMC.3 adapter card (P/N 80172)....32
Figure 10. Innovative PCI 64/66 – XMC.3 (4x lanes) adapter card (P/N 80167-0)....32
Figure 11. Innovative x8 Lane PCI Express – XMC.3 (8x lanes) adapter card (P/N 80173-0)....32
Figure 12. Innovative Compact PCI/PXI – XMC.3 (x4 lanes) adapter card (P/N 80207-0)....33
Figure 13. eInstrument Node – cabled PCI Express adapter (x1 lane) for XMC Modules (II P/N 90181)....33
Figure 14. eInstrument PC – embedded PC (Windows/Linux) hosts two XMC modules (II P/N 90199)....33
Figure 15. X6 XMC Family Block Diagram....37
Figure 16. PCI Express Interface Block Diagram....40
Figure 17. PCI Interface Block Diagram....41
Figure 18. X6 DRAM Interface....43
Figure 19. X6 Digital IO Block Diagram....46
Figure 20. Default Digital IO Configuration....48
Figure 21. X6 Aurora Ports on P16....52
Figure 22. X6 Module Cluster Using Aurora....53
Figure 23. X6 Temperature Monitoring....55
Figure 24. Temperature Sensor Location....56
Figure 25. Example Fan Control Circuit....58
Figure 26. X6 Conduction Cooling Thermal Connection Areas....59
Figure 27. X6 Heat Spreader for FPGA....60
Figure 28. eInstrument Node Enclosure (P/N 90181) Supports Standalone Operation....65
Figure 29. X6 Logic Configuration Subsystem....66
Figure 30. X6 Configuration FLASH Programmer....67
Figure 31. X6-RX Module....109
Figure 32. X6-RX Block Diagram....110
Figure 33. X6-RX A/D Channel Front End....112
Figure 34. Sample Clock Generation and Distribution Block Diagram....113
Figure 35. Input Clock/Reference Electrical Specifications....114
Figure 36. SNAP Example Sample Clock Controls....115
Figure 37. Clock Path Using External Reference Input....118
Figure 38. Clock Path Using External Clock Input....119
Figure 39. Analog Triggering Timing....122
Figure 40. X6-RX SNAP Example Triggering Controls....124
Figure 41. X6-RX FrameWork Logic Data Flow....126
Figure 42. Typical Performance Evaluation Setup....131

Figure 43. Connectors J1-J6 Functions....140
Figure 44. P15 XMC Connector Orientation....141
Figure 45. P16 XMC Connector Orientation....144
Figure 46. PMC JN1 Connector Schematic....147
Figure 47. PMC JN2 Connector Schematic....150
Figure 48. PMC JN4 Connector Schematic....153
Figure 49. X6-RX JP3 Orientation, board face view....155
Figure 50. X6-RX JP3 Orientation, board top edge view....155
Figure 51. X6-RX Mechanicals (Top View)....158
Figure 52. X6-RX Mechanicals (Bottom View)....158

Chapter 1: *Introduction*

Real Time Solutions!

Thank you for choosing Innovative Integration, we appreciate your business! Since 1988, Innovative Integration has grown to become one of the world's leading suppliers of DSP and data acquisition solutions. Innovative offers a product portfolio unrivaled in its depth and its range of performance and I/O capabilities .

Whether you are seeking a simple DSP development platform or a complex, multiprocessor, multichannel data acquisition system, Innovative Integration has the solution. To enhance your productivity, our hardware products are supported by comprehensive software libraries and device drivers providing optimal performance and maximum portability.

Innovative Integration's products employ the latest digital signal processor technology thereby providing you the competitive edge so critical in today's global markets. Using our powerful data acquisition and DSP products allows you to incorporate leading-edge technology into your system without the risk normally associated with advanced product development. Your efforts are channeled into the area you know best ... your application.

Vocabulary

What is X6-RX?
The X6-RX is a flexible receiver that integrates IF digitizing with signal processing on a PMC IO module. The module provides up to 24 configurable receiver channels with a powerful Xilinx Virtex 6 FPGA signal processing core, and high performance PCI Express/PCI host interface. With the X6-RX, IF recorders can log both the digitized raw data and channels real-time sustaining rates over 2 GB/s.

The X6-RX features four, 16-bit 160 MSPS A/Ds with digital downconverter (DDC). IF frequencies of up to 300 MHz are supported. The sample clock is from either a low-jitter PLL or external input. Multiple cards can be synchronized for sampling and downconversion.

A Xilinx Virtex6 LX240T (SX315T and SX475T available) with 4 banks of 1GB DRAM provide a very high performance DSP core with over 2000 MACs (SX315T). The close integration of the analog IO, memory and host interface with the FPGA enables real-time signal processing at extremely high rates.

The onboard DDC ASIC device, connected directly to the FPGA, provides up to 24 narrowband or 8 wideband channels with input from two A/D channels. The DDC performs complex or real downconversion, with flexible controls for mixing, filtering, decimation, output formats and data rates. Channels can be synchronized to support beam forming or frequency hopped systems.

The X6 family power is less than 15W for typical operation. VITA 20 conduction cooling is used with a heat-spreader and sink. Ruggedization levels for wide-temperature operation and conformal coating are supported.

The FPGA logic can be fully customized using VHDL and MATLAB using the Frame Work Logic toolset. The MATLAB BSP supports real-time hardware-in-the-loop development using the graphical, block diagram Simulink environment with Xilinx System Generator. IP cores for DDC, demodulation, and FFT are available.

Software tools for host development include C++ libraries and drivers for Windows and Linux. Application examples demonstrating the module features and use are provided.

This extremely versatile module is easily adapted for use in virtually any type of system. Our XMC carrier adapters offer conduction and convection cooling and are available for a range of interfaces including Desktop PCI, Desktop PCI Express, Cabled PCI Express, CompactPCI, and PXI/PXI Express. This module is also readily installed into Innovative Integration's eInstrument Embedded PC, SBC-ComEx Single-Board Computer, and Andale Data Loggers.

What is Malibu?
Malibu is the Innovative Integration-authored component suite, which combines with the Borland, Microsoft or GNU C++ compilers and IDEs to support programming of Innovative hardware products under Windows and Linux. Malibu supports both high-speed data streaming plus asynchronous mailbox communications between the DSP and the Host PC, plus a wealth of Host functions to visualize and post-process data received from or to be sent to the target DSP.

What is C++ Builder?
C++ Builder is a general-purpose code-authoring environment suitable for development of Windows applications of any type. Armada extends the Builder IDE through the addition of functional blocks (VCL components) specifically tailored to perform real-time data streaming functions.

What is DialogBlocks?
DialogBlocks is an easy-to-use dialog editor for your wxWidgets applications, generating C++ code and XRC resource files. Using sizer-based layout, DialogBlocks helps you build dialogs and panels that look great on Windows, Linux or any supported wxWidgets platform. Add context-sensitive help text, tooltips, images, splitter windows and more.

What is wxWidgets?
wxWidgets was started in 1992 by Julian Smart at the University of Edinburgh. Initially started as a project for creating applications portable across Unix and Windows, it has grown to support the Mac platform, WinCE, and many other toolkits and platforms. The number of developers contributing to the project is now in the dozens and the toolkit has a strong userbase that includes everyone from open source developers to corporations such as AOL. So what is special about wxWidgets compared with other cross-platform GUI toolkits?

wxWidgets gives you a single, easy-to-use API for writing GUI applications on multiple platforms that still utilize the native platform's controls and utilities. Link with the appropriate library for your platform (Windows/Unix/Mac, others coming shortly) and compiler (almost any popular C++ compiler), and your application will adopt the look and feel appropriate to that platform. On top of great GUI functionality, wxWidgets gives you: online help, network programming, streams, clipboard and drag and drop, multithreading, image loading and saving in a variety of popular formats, database support, HTML viewing and printing, and much more.

What is Qt?
Qt (pronounced officially as "cute (KYOOT)" although commonly known also as "Q.T. (KYOO-TEE)") is a cross-platform application development framework widely used for the development of GUI programs (in which case it is known as a widget toolkit), and also used for developing non-GUI programs such as console tools and servers. Qt is most notably used in Google Earth, KDE, Opera (before 10.60 version), OPIE, Skype, VLC media player and VirtualBox. It is produced by Nokia's Qt Development Frameworks division, which came into being after Nokia's acquisition of the Norwegian company Trolltech, the original producer of Qt, on June 17, 2008.

Qt uses standard C++ but makes extensive use of a special pre-processor (called the Meta Object Compiler, or moc) to enrich the language. Qt can also be used in several other programming languages via language bindings. It runs on all major platforms and has extensive internationalization support. Non-GUI features include SQL database access, XML parsing, thread management, network support, and a unified cross-platform API for file handling.

Distributed under the terms of the GNU Lesser General Public License (among others), Qt is free and open source software. All editions support a wide range of compilers, including the GCC C++ compiler and the Visual Studio suite..

So what is special about wxWidgets compared with other cross-platform GUI toolkits? Qt gives you a single, easy-to-use API for writing GUI applications on multiple platforms that still utilize the native platform's controls and utilities. Link with the appropriate library for your platform (Windows/Unix/Mac) and compiler (GNU C++), and your application will adopt the look and feel appropriate to that platform. On top of great GUI functionality, Qt gives you: online help, network programming, streams, clipboard and drag and drop, multithreading, image loading and saving in a variety of popular formats, database support, HTML viewing and printing, and much more.

What is Microsoft MSVC?
MSVC is a general-purpose code-authoring environment suitable for development of Windows applications of any type. Armada extends the MSVC IDE through the addition of dynamically created MSVC-compatible C++ classes specifically tailored to perform real-time data streaming functions.

What kinds of applications are possible with Innovative Integration hardware?
Data acquisition, data logging, stimulus-response and signal processing jobs are easily solved with Innovative Integration baseboards using the Malibu software. There are a wide selection of peripheral devices available in the Matador DSP product family, for all types of signals from DC to RF frequency applications, video or audio processing. Additionally, multiple Innovative Integration baseboards can be used for a large channel or mixed requirement systems and data acquisition cards from Innovative can be integrated with Innovative's other DSP or data acquisition baseboards for high-performance signal processing.

Why do I need to use Malibu with my Baseboard?
One of the biggest issues in using the personal computer for data collection, control, and communications applications is the relatively poor real-time performance associated with the system. Despite the high computational power of the PC, it cannot reliably respond to real-time events at rates much faster than a few hundred hertz. The PC is really best at processing data, not collecting it. In fact, most modern operating systems like Windows are simply not focused on real-time performance, but rather on ease of use and convenience. Word processing and spreadsheets are simply not high-performance real-time tasks.

The solution to this problem is to provide specialized hardware assistance responsible solely for real- time tasks. Much the same as a dedicated video subsystem is required for adequate display performance, dedicated hardware for real-time data collection and signal processing is needed. This is precisely the focus of our baseboards – a high performance, state-of-the-art, dedicated digital signal processor coupled with real-time data I/O capable of flowing data via a 64-bit PCI bus interface.

The hardware is really only half the story. The other half is the Malibu software tool set which uses state of the art software techniques to bring our baseboards to life in the Windows environment. These software tools allow you to create applications for your baseboard that encompass the whole job - from high speed data acquisition, to the user interface.

Finding detailed information on Malibu
Information on Malibu is available in a variety of forms:

- Data Sheet (http://www.innovative-dsp.com/products/malibu.htm)
- On-line Help
- Innovative Integration Technical Support

- Innovative Integration Web Site (www.innovative-dsp.com)

Online Help

Help for Malibu is provided in a single file, Malibu.chm which is installed in the Innovative\Documentation folder during the default installation. It provides detailed information about the components contained in Malibu - their Properties, Methods, Events, and usage examples. An equivalent version of this help file in HTML help format is also available online at http://www.innovative-dsp.com/support/onlinehelp/Malibu.

Innovative Integration Technical Support

Innovative includes a variety of technical support facilities as part of the Malibu toolset. Telephone hotline supported is available via

Hotline (805) 578-4260 8:00AM-5:00 PM PST.

Alternately, you may e-mail your technical questions at any time to:

techsprt@innovative-dsp.com.

Also, feel free to register and browse our product forums at http://forum.iidsp.com/, which are an excellent source of FAQs and information submitted by Innovative employees and customers.

Innovative Integration Web Site

Additional information on Innovative Integration hardware and the Malibu Toolset is available via the Innovative Integration website at www.innovative-dsp.com

Typographic Conventions

This manual uses the typefaces described below to indicate special text.

Typeface	Meaning
`Source Listing`	Text in this style represents text as it appears onscreen or in code. It also represents anything you must type.
Boldface	Text in this style is used to strongly emphasize certain words.
Emphasis	Text in this style is used to emphasize certain words, such as new terms.
Cpp Variable	Text in this style represents C++ variables
Cpp Symbol	Text in this style represents C++ identifiers, such as class, function,

	or type names.
KEYCAPS	Text in this style indicates a key on your keyboard. For example, “Press ESC to exit a menu”.
Menu Command	Text in this style represents menu commands. For example “Click View | Tools | Customize”

Chapter 2: *Windows Installation*

This chapter describes the software and hardware installation procedure for the Windows platform (WindowsXP, Vista, and Windows 7).

> **Do *NOT*** install the hardware card into your system at this time. This will follow the software installation.

Host Hardware Requirements

The software development tools require an IBM or 100% compatible Pentium IV - class or higher machine for proper operation. An Intel-brand processor CPU is *strongly recommended*, since AMD and other "clone" processors are not guaranteed to be compatible with the Intel MMX and SIMD instruction-set extensions which the Armada and Malibu Host libraries utilize extensively to improve processing performance within a number of its components. The host system must have at least 1 GB of memory (2 GB recommended), 1 GB available hard disk space, and a DVD-ROM drive. Most versions of Windows released after Win2000 including XP, Vista, or Windows 7 (referred to herein simply as *Windows*) or later is required to run the developer's package software, and are the target operating systems for which host software development is supported.

Software Installation

The development package installation program will guide you through the installation process.

> **Note:** Before installing the host development libraries (VCL components or MFC classes), you must have Microsoft MSVC Studio (version 9 or later), CodeGear RAD Studio 2007/2009, Embarcadero Rad Studio 2010 or QtCreator installed on your system, depending on which of these IDEs you plan to use for Host development. If you are planning on using these environments, it is imperative that they are tested and known-operational before proceeding with the library installation. If these items are not installed prior to running the Innovative Integration install, the installation program will not permit installation of the associated development libraries. However, drivers and DLLs may be installed to facilitate field deployment.

You must have **Administrator Privileges** to install and run the software/hardware onto your system, refer to the Windows documentation for details on how to get these privileges.

Starting the Installation

To begin the installation, start Windows. Shut down all running programs and disable anti-virus software. Insert the installation **DVD**. If Autostart is enabled on your system, the install program will launch. If the DVD does not Autostart, click on Start | Run... Enter the path to the **Setup.bat** program located at the root of your DVD-ROM drive (i.e. **E:\Setup.bat**) and click "OK" to launch the setup program.

> **SETUP.BAT** detects if the OS is 64-bit or 32-bit and runs the appropriate installation for each environment. It is important that this script be run to launch an install.

When installing on a Vista OS, the dialog below may pop up. In each case, select "Install this driver software anyway" to continue.

Windows Security
Windows can't verify the publisher of this driver software
Don't install this driver software
You should check your manufacturer's website for updated driver software for your device.
Install this driver software anyway
Only install driver software obtained from your manufacturer's website or disc. Unsigned software from other sources may harm your computer or steal information.
See details

Figure 1. Vista Verification Dialog

The Installer Program

After launching Setup, you will be presented with the following screen.

Figure 2. Innovative Install Program

Using this interface, specify which product to install, and where on your system to install it.

1) Select the appropriate product from the Product Menu.

2) Specify the path where the development package files are to be installed. You may type a path or click "**Change**" to browse for, or create, a directory. If left unchanged, the install will use the default location of "C:\Innovative".

3) Typically, most users will perform a "Full Install" by leaving all items in the "**Components to Install**" box checked. If you do not wish to install a particular item, simply uncheck it. The Installer will alert you and automatically uncheck any item that requires a development environment that is not detected on your system.

4) Click the Install button to begin the installation.

Note: The default “Product Filter” setting for the installer interface is “Current Only” as indicated by the combo box located at the top right of the screen. If the install that you require does not appear in the “Product Selection Box” (1), Change the “Product Filter” to “Current plus Legacy”.

Each item of the checklist in the screen shown above, has a sub-install associated with it and will open a sub-install screen if checked. For example, the first sub-install for “Quadia - Applets, Examples, Docs, and Pismo libraries” is shown below.

The installation will display a progress window, similar to the one shown below, for each item checked.

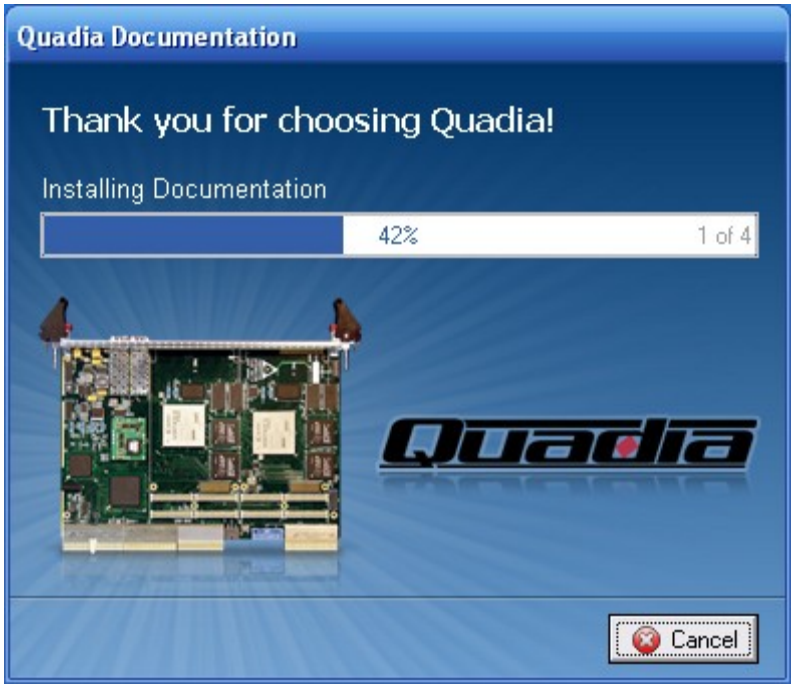
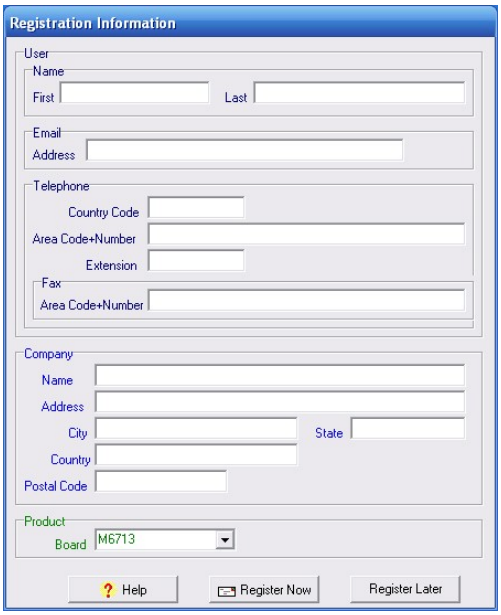


Figure 3. Progress is shown for each section.

Tools Registration



At the end of the installation process you will be prompted to register. If you decide that you would like to register at a later time, click "Register Later".

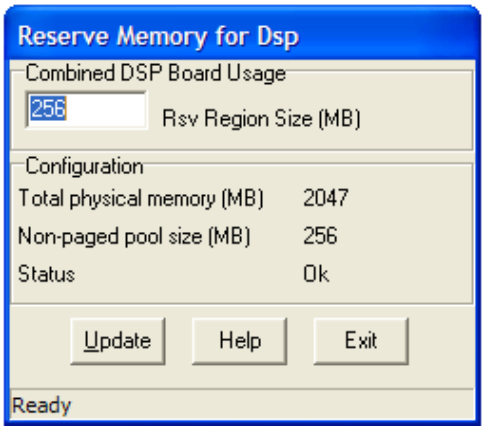
When you are ready to register, click Start | All Programs | Innovative | <Board Name> | Applets. Open the New User folder and launch **NewUser.exe** to start the registration application. The registration form to the left will be displayed.

Before beginning DSP and Host software development, you must register your installation with Innovative Integration. Technical support will not be provided until registration is successfully completed. Additionally, some development applets will not operate until unlocked with a passcode provided during the registration process.

It is recommend that you completely fill out this form and return it to Innovative Integration, via email or fax. Upon receipt, Innovative Integration will provide access codes to enable technical support and unrestricted access to applets.

Figure 4. ToolSet registration form

Bus Master Memory Reservation Applet.



At the conclusion of the installation process, **ReserveMem.exe** will run (except for SBC products). This will allow you to set the memory size needed for the busmastering to occur properly. This applet may be run from the start menu later if you need to change the parameters.

For optimum performance, reserve at least 64 MB of memory for each Innovative board to be used simultaneously within the PC plus 32 MB for other system use. For example, if using two X5-400M modules, reserve 2 * 64 + 32 MB = 160 MB. To reserve this memory, the registry must be updated using the ReserveMem applet. Simply type the desired size into the Rsv Region Size (MB) field, click **Update** and the applet will update the registry for you. If at any time you change the number of boards in your system, then you must invoke this applet found in Start | All Programs | Innovative | <target board> | Applets | Reserve Memory.

After updating the system exit the applet by clicking the **exit** button to resume the installation process.

Figure 5. BusMaster configuration

At the end of the install process, the following screen will appear.

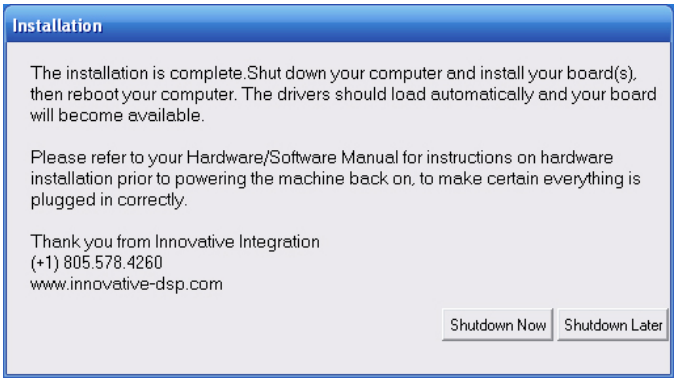

Installation
The installation is complete.Shut down your computer and install your board(s), then reboot your computer. The drivers should load automatically and your board will become available.
Please refer to your Hardware/Software Manual for instructions on hardware installation prior to powering the machine back on, to make certain everything is plugged in correctly.
Thank you from Innovative Integration
(+1) 805.578.4260
www.innovative-dsp.com
Shutdown Now
Shutdown Later

Figure 6. Installation complete

Click the "Shutdown Now" button to shut down your computer. Once the shutdown process is complete unplug the system power cord from the power outlet and proceed to the next section, "Hardware Installation."

Hardware Installation

Now that the software components of the Development Package have been installed the next step is to configure and install your hardware. Detailed instructions on board installation are given in the Hardware Installation chapter, following this chapter.

> **IMPORTANT:** Many of our high speed cards, especially the PMC and XMC Families, require forced air from a fan on the board for cooling. Operating the board without proper airflow may lead to improper functioning, poor results, and even permanent physical damage to the board. These boards also have temperature monitoring features to check the operating temperature. The board may also be designed to intentionally fail on over-temperature to avoid permanent damage. See the specific hardware information for airflow requirements.

After Power-up

After completing the installation, boot your system into Windows.

Innovative Integration boards are plug and play compliant, allowing Windows to detect them and auto-configure at start-up. Under rare circumstances, Windows will fail to auto-install the device-drivers for the JTAG and baseboards. If this happens, please refer to the "TroubleShooting" section.

Installation on a Deployed System

The above instructions install the complete development platform onto a system for the development of application software. Often, however, a developed application needs to be installed on a system that will only be used to run the program. In this instance, installing the complete library is overkill.

To support this situation, Innovative has a minimal installation program called "MalibuRED". This is short for Malibu Redistributable. This install will install the driver software and support DLLs required to run a Malibu application.

> **Note:** Specific applications may have their own, additional requirements that are not covered by MalibuRED. For example, .NET applications require the .NET libraries to be installed as well. Installation programs for .NET can be obtained from Microsoft over the Internet.

Running MalibuRed

MalibuRED can be found on the installation CD in the Windows-32\Malibu subdirectory. The name of the installation file is MalibuRED.exe. Running the program displays the setup screen for the installer:

Select your baseboard...
X3-A4D4
Go
Cancel
NOTE: This install must be completed once for each individual product on systems utilizing multiple Innovative Integration baseboards.

Using the combo box, select the appropriate baseboard to install support for. In this case, we are installing an X3-A4D4 board. If support for multiple cards is needed, the program must be run to completion once for each type of board. This is required because parts of the installation, such as baseboard device drivers, may be different for different board types.

After selecting the board, press "Go" to begin installation. The window changes to display the progress of the install.

Windows Installation

After completing the installation, reboot the system to allow Windows to recognize the new drivers. Then proceed with the Hardware Installation as in the development system installation above.

Chapter 1: *Installation on Linux*

This chapter contains instruction on the installation of the baseboard software for Linux operating systems.

Software installation on Linux is performed by loading a number of **packages**. A Package is a special kind of archive file that contains not only the files that are to be installed, but also installation scripts and dependency information to allow a smooth fit into the system. This information allows the package to be removed, or patched. Innovative uses RPM packages in its installs.

Package File Names
A package file name such as Malibu-LinuxPeriphLib-1.1-3.i586.rpm encodes a lot of information.

Package Name		Package ID		Information Fields	
Distribution	**Subpackage**	**Version**	**Revision**	**Hardware Type**	**Extension**
Malibu-Linux	PeriphLib	1.1	3	i586	.rpm

Prerequisites for Installation

In order to properly use the baseboard example programs and to develop software using the baseboard, some packages need to be installed before the actual baseboard package.

The Redistribution Package Group - MalibuRed

This set of packages contain the libraries and drivers needed to run a program using Malibu. This group is called "MalibuRed" because it contains the packages needed to allow running Malibu based programs on a target, non-development machine. (Red is short for 'redistributable').

MalibuRed Packages	Description
WinDriver-9.2-1.i586.rpm	Installs WinDriver 9.2 release.
MalibuLinux-Red-**[ver]**-**[rel]**.i586.rpm	Installs Baseboard Driver Kernel Plugin.
intel-ipp_rti-5.3p.x32.rpm	Installs Intel IPP library redistributable files.

The installation CD, or the web site contains a file called **LinuxNotes.pdf** giving instructions on how to load these packages and how to install the drivers onto your Linux machine. This file is also loaded onto the target machine by the the Malibu-LinuxRed RPM. These procedures need to be completed for every target machine.

Malibu

To develop software for a baseboard the Malibu packages also must be installed.

Malibu Packages	Description
Malibu-LinuxPeriphLib-**[ver]-[rel]**.i586.rpm	Installs Malibu Source, Libraries and Examples.

Other Software

Our examples use the DialogBlocks designer software and wxWidgets GUI library package for user interface code. If you wish to rebuild the example programs you will have to install this software as well.

Package	Company	URL
wxWidgets	wxWidgets	http://www.wxwidgets.org
DialogBlocks	Anthemion	http://www.anthemion.co.uk.org/dialogblocks

Baseboard Package Installation Procedure

Each baseboard installation for Linux consists of one or more package files containing self-extracting packages of compressed files, as listed in the table below. Note that package version codes may vary from those listed in the table.

Each of these packages automatically extract files into the `/usr/Innovative` folder, herein referred to as the *Innovative root folder* in the text that follows. For example, the X5-400 RPM extracts into `/usr/Innovative/X5-400-[ver]`. A symbolic link named `X5-400` is then created pointing to the version directory to allow a single name to apply to any version that is in use.

Board Packages

Baseboard	Packages	Description
X5-400M	Malibu-LinuxPeriphLib-[**ver]-[rel]**.i586.rpm	Board files and examples.
X5-210M	X5-210M-LinuxPeriphLib-**[ver]-[rel]**.i586.rpm	Board files and examples.
X3-10M	X3-10M-LinuxPeriphLib-**[ver]-[rel]**.i586.rpm	Board files and examples.
X3-25M	X3-25M-LinuxPeriphLib-**[ver]-[rel]**.i586.rpm	Board files and examples.
X3-A4D4	X3-A4D4-LinuxPeriphLib-**[ver]-[rel]**.i586.rpm	Board files and examples.
X3-SD	X3-SD-LinuxPeriphLib-**[ver]-[rel]**.i586.rpm	Board files and examples.
X3-SDF	X3-SDF-LinuxPeriphLib-**[ver]-[rel]**.i586.rpm	Board files and examples.
X3-Servo	X3-Servo-LinuxPeriphLib-**[ver]-[rel]**.i586.rpm	Board files and examples.
SBC-ComEx	Sbc-ComEx-LinuxPeriphLib-**[ver]-[rel]**.i586.rpm	Board files and examples.

Unpacking the Package

As root, type:

```
rpm -i -h X5-400-LinuxPeriphLib-1.1-4.i586.rpm
```

This extracts the X5-400 board files into the Innovative root directory. Use the package for the particular board you are installing.

Creating Symbolic Links

The example programs assume that the user has created symbolic links for the installed board packages. A script file is provided to simplify this operation by the Malibu Red package. In the MalibuRed/KerPlug directory, there is a script called quicklink.

```
quicklink X5-400 1.1
```

These commands will create a symbolic link `X5-400` pointing to `X5-400-1.1`.

This script can be moved to the user's `bin` directory to allow it to be run from any directory.

Completing the Board Install

The normal board install is complete with the installation of the files. The board driver install is already complete with the loading of the Malibu Red package. If there are any board-specific steps they will be listed at the end of this chapter.

Linux Directory Structure

When a board package is installed, its files are placed under the `/usr/Innovative` folder. The base directory is named after the board with a version number attached -- for example, the version 2.0 X5-400 RPM extracts into `/usr/Innovative/X5-400-2.0`.

This allows multiple version of installs to coexist by using a symbolic link to point to a particular version. Changing the symbolic link changes with version will be used.

Under the main directory there are a number of subdirectories.

Applets
The applets subdirectory contains small application programs that aid in the use of the board. For example, there is a Finder program that allows the user to flash an LED on the board to determine which board is associated with a target number. See the Applets chapter for a fuller description of the applets for a board.

Documentation
This directory contains any documentation files for the project. Open the index.html file in the directory with a web browser to see the available files and a description of the contents.

Examples
This directory and its subdirectories contain the projects, source and example programs for the board.

Hardware
This directory contains files associated with programming the board Logic and any logic images provided.

Chapter 2: *Hardware Installation*

The X6 XMC cards may be used on a variety of host cards supporting an XMC.3 PCI Express (VITA standard 42.3 compatible) site. XMC P16 is also used for system integration including use as a dedicated data channel.

Compatible Host Cards

X6 XMC cards are compatible with the VITA 42.3 PCI Express mezzanine module sites. The card form factor is IEEE 1386 with XMC connectors P15 and P16. P15 is used for PCI Express, while P16 is used as digital IO unique to the X6 modules. The X6 modules mount 10mm from the host card and may use standoffs for mechanically securing the module to the host.

PCI Express Bus Requirement	Type
Standard	PCI Express 1.0a
Lanes	8
Bus Speed	Gen1: 2.5 Gbps per lane per direction Gen2: 5 Gbps per lane per direction
Power Supplies	3.3V and VPWR (+5V or +12V) is required on all XMCs.

Table 1. X6 XMC Bus Requirements

Note: PCI Express interfaces supports Gen1 (2.5 Gbps) or Gen2 (5 Gbps) operation. For Gen2, the host system must support Gen2 operation in the installed site. For Gen2 x8, the X6 module must be ordered with -2 FPGA speed grade or better.

Adapter cards for XMC modules, shown below, allow X6 XMC modules to be used in a desktop PCs with PCI Express or PC slots.

To get out of the PC chassis, consider using either the eInstrument Node. The eInstrument node can host one XMC using cabled PCI Express using a cable as long as 5 meters.

For an intelligent computing node, the eInstrument PC provides an embedded PC running Windows or Linux, disk drives, and other PC peripherals with two XMC module sites.

Preferred for X6 Modules

Figure 7. Innovative x8 lane PCIe – XMC.3 adapter card (P/N 80259)

Figure 8. Innovative 3U VPX- XMC.3 adapter card (P/N 80260)

Conduction-cooled or Air-cooled versions are offered.

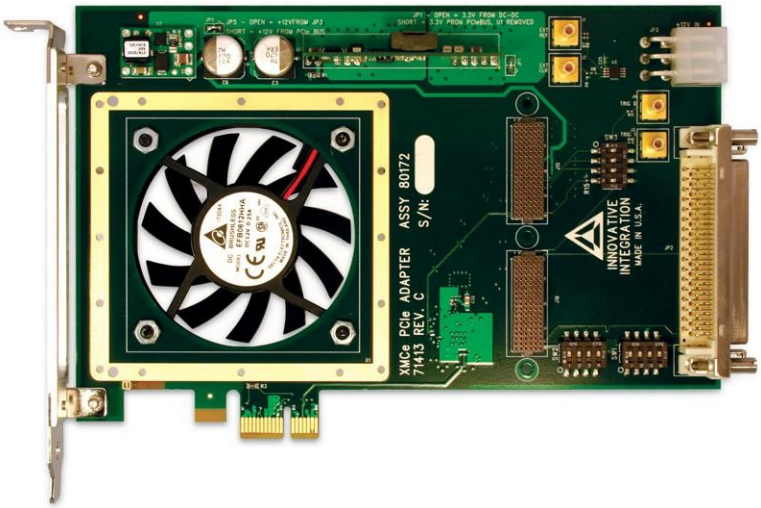


Figure 9. Innovative Single lane PCIe – XMC.3 adapter card (P/N 80172)

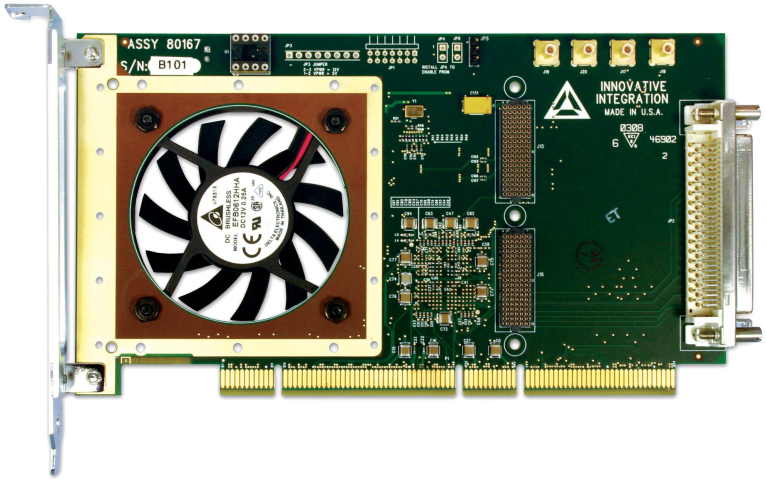


Figure 10. Innovative PCI 64/66 – XMC.3 (4x lanes) adapter card (P/N 80167-0)

Figure 11. Innovative x8 Lane PCI Express – XMC.3 (8x lanes) adapter card (P/N 80173-0)

Figure 12. Innovative Compact PCI/PXI – XMC.3 (x4 lanes) adapter card (P/N 80207-0)

Figure 13. eInstrument Node – cabled PCI Express adapter (x1 lane) for XMC Modules (II P/N 90181)

Figure 14. eInstrument PC – embedded PC (Windows/Linux) hosts two XMC modules (II P/N 90199)

XMC systems should be be compatible with VITA 42.3 specification for P15.

The P16 interface can only be used when the PCI Express interface is present and active. The XMC P16 interface to the host may be customized in the Application Logic. Host card support for P16 interfaces varies so this must be verified on a case-by-case basis to determine compatibility. In the specific description of each module, the P16 connection pinout is provided in this manual. Innovative's adapter cards provide access to P16 IO for system integration. Contact technical support if you need any assistance in checking compatibility with the X6 XMC P16 interface.

System Requirements

The PCI Express slot must be PCI Express 1.0a compatible and be able to map the following resources.

Table 2. Required PCIe Resource Allocations

Required PCIe Resources
BAR0 : 1MB Memory
BAR1 : 64K B Memory
1 Interrupt

When you first plug in the module, it will then be found and the driver should be installed for the module. The standard driver resides in the \Innovative\drivers directory after software installation.

The XMC module may be used an any PCI Express slot supporting 1 or more lanes. Innovative offers a line of adapters for use with the X6 series which allow the module to be used in PCI and cPCI environments. These adapters may limit the number of lanes available to the X6 for host communication, depending on their design.

Power Considerations

Each XMC should be reviewed for its specific power, cooling and any special mechanical considerations. For each module the power consumption and required power supplies are shown in the specific discussion for that module.

For desktop applications, you MUST provide forced air cooling with 5-10 CFM capability. The air must blow directly on the Virtex 6 device.

The X6 XMCs are designed to operate over the typical commercial temperature range of 0 to 70 C, but this relies on sufficient forced air cooling for most installations and modules. At the lower temperatures, it is also required that the environment be non-condensing for the standard commercial modules. Extended temperature versions of many modules are available with conformal coating if the environment is more demanding.

During operation, the module temperature should be monitored to prevent unexpected shutdown. If the module temperature exceeds 85C (L0 rating), the module will deactivate on-board power supplies to avoid overheating. Please refer to the section on thermal properties for more details.

PCI Express slots are rated for their power capability. Each installation should be reviewed to verify that the host card plus the module do not exceed the rated power of the slot. The power consumption for each module is provided in the specific discussion of that module. Most slots support 15W in desktop systems and provide 3.3V and 12V. Some XMC modules require -12V, which must be provided by the host card.

Mechanical Considerations

The X6 modules conform to IEEE1386 CMC specification and ANSI/VITA 42.0 specifications, except as specified on individual modules. These specifications define the size of the module and mounting requirements. In short the modules are 75 x 150 mm and mount 10 mm from the host card. This allows the XMC modules to fit in one slot in desktop PC or compact PCI systems.

For ruggedness, the modules should be mounted to a host card using the mounting screws and standoffs. The host bracket should securely hold the XMC front bracket so that the module is snug in the bracket opening. This reduces mechanical strain on the XMC connectors from the front panel cable attachments.

The card may be secured to the host card with two standoffs and four screws as shown. The Digikey number is provided (www.digikey.com) for convenience; many suppliers have these standard screw and standoff sizes.

For high vibration applications, screws should be mounted with locking compound such as Loctite 222.

Table 3. XMC Mounting Hardware

Description	Quantity	Digikey Part Number
Metric pan head screw, M3X6, 3mm x 5mm	5	H742-ND
Threaded Standoff, 10mm with 3mm thread	3	4391K

During the logic and firmware design process, some modules require access to JTAG connectors for use with the development tools. This may require greater access space than the final installation. A variety of engineering tools are available to assist the designer during the development process such as PCIe to XMC adapters. A cabled version allows the module to operate outside the chassis. An open chassis may also be required to get access to the XMC during the development process so that the cables are easily accessible.

Chapter 3: *About the X6 XMC Modules*

In this chapter, we will discuss the common features of the X6 module family. Specifics on each module are covered in later chapters.

X6 XMC Architecture

The X6 XMC modules share a common architecture and many features such as the PCI/PCI Express interface, data buffering features, the Application Logic, and other system integration features. This allows the X6 XMC modules to utilize common software and logic firmware, while providing unique analog and digital features.

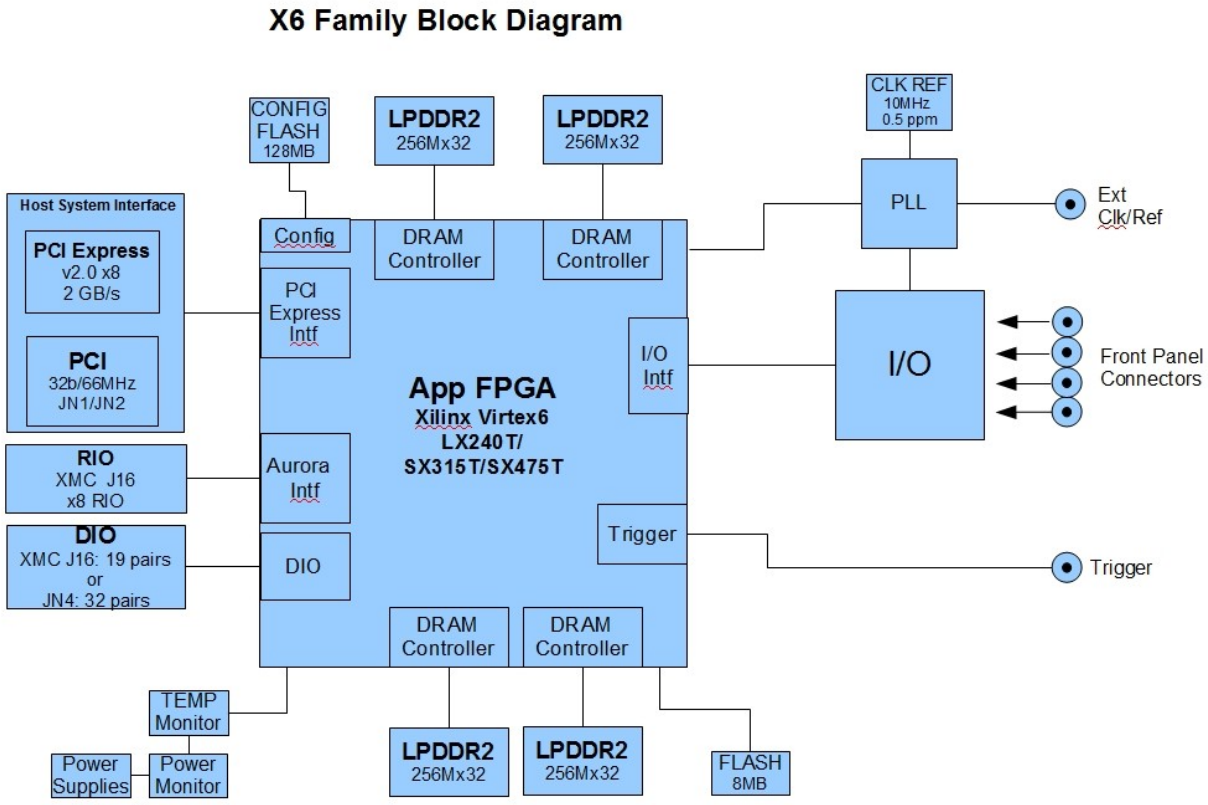


Figure 15. X6 XMC Family Block Diagram

The X6 XMCs have a variety of analog and digital IO front ends suited to many applications.

Table 4. X6 XMC Family

X6 XMC	Features	Applications
X6-RX	4 A/D channels, 16bit, 160 MHz, integrated 4 channel DDC	IF digitizing and receivers.
X6-400M	2 channels 14-bit, 400 MSPS A/D and 2 channels 16-bit, 500 MSPS DAC (single channel at 1 GHz)	Wireless transceiver, IF digitizing, RADAR, Electronic Warfare
X6-COP	2 10Gb Ethernet ports; SFP+ support fiber optic connections	Sensor networks, Ethernet monitoring, remote radio heads, coprocessing
X6-TX	4 DAC channels, 16-bit, 1 GSPS each, pattern generation mode for DRAM	Arbitrary waveform generation, communications waveform generation
X6-GSPS	2 A/D channels, 12-bit, at 1.8GSPS or single channel at 3.6 GPSP	RADAR, communications receivers, pulse digitizing
X6-COP	SX315T/SX475T/LX550T FPGA and 6GB memory	Coprocessing

Table 5. X6 XMC Family

The X6 XMCs feature a Xilinx Virtex-6 SX315T, SX475 or LX240T core for signal processing and control. (Early release products are LX240T). In addition to the features in the Virtex-6 logic such as embedded multipliers and memory blocks, the FPGA computing core has four independent banks of LPDDR2 DRAM for data buffering and computing memory.

There are also a number of support peripherals for IO control and system integration. Each XMC may have additional application-specific support peripherals.

Peripheral	Features
XMC.3 PCI Express interface	The X6 cards have an x8 lane PCI Express interface supporting Gen1 or Gen 2 operation. Gen x8 requires speed upgrade to FPGA. The XMC.3 host interface integrates with PCI Express systems using eight lanes operating at 5 Gbps (Gen2) or 2.5 Gbps (Gen1) each that provide up to 8 GBytes/sec data rate on the bus (full duplex). This interface complies with VITA standard 42.3 which specifies PCI Express interface for the XMC module format. The PCI Express interface is not available when PCI interface is being used. Data transfers between the X6 and host use the Velocia packet system. This protocol automates the DMA transfers using a credit-based flow control for the packet transfers with the host. The system is extensible through the use of virtual channels defined in the logic design. A secondary command channel provides independent interface for control and status outside of the data channel that is extensible to custom applications.
PCI Interface	The module implements a PCI bus interface using PMC connectors JN1 and JN2. When the PCI interface is used, the X6 module is compatible with PMC module sites. The PCI interface is a 32-bit, 66MHz interface capable of up to 264 MB/s transfer rates and conforms to PCI Local Bus Specification

Peripheral	Features
	3.0. Voltage standard is 3.3V with 5V IO tolerance. The PCI Express interface is not available when PCI interface is being used. Data transfers between the X6 and host employ the Velocia packet system. This protocol automates the DMA transfers using a credit-based flow control for the packet transfers with the host. The ssytem is extensible through the use of virtual channels defined in the logic design. A secondary command channel provides independent interface for control and status outside of the data channel that is extensible to custom applications.
XMC J16 or JN4 DIO	Digital IO is provided that has direct connection to the FPGA. The card can be configured so that these digital IO are available on either J16 or JN4 (factory installed option). J16 option : 19 digital IO pairs (38 signals); LVCMOS 2.5V or LVDS JN4 option : 32 digital IO pairs (64 signals); LVCMOS 2.5V or LVDS
XMC P16 Communications Ports	Eight high speed serial links (GTX) operating at up to 5 Gbps each are available. The Framework Logic implements either Aurora ports for system integration.
Timing and triggering	Flexible clocking and synchronization features for I/O including front panel clock or reference, system reference and PPS inputs through J16 connector.
Data buffering and Computational Memory	Four 1GB LPDDR2 DRAM devices are used provide data buffering, processor memory and computation memory for the Application FPGA
Alert Log	Monitors system events and error conditions to help manage the data acquisition process
Temperature and Power Monitoring	Autonomous temperature and power monitoring; over-temperature protection for the module shuts down power; temperature reporting and alerts for monitoring; power supply health monitoring.

Table 6. X6 XMC Family Peripherals

X6 Computing Core

The X6 XMC module family has an FPGA-based computing core that controls the data acquisition process, provides data buffing and host communications. The computing core consists of a Xilinx Virtex-6FPGA, one bank of DDR2 DRAM (4Gbits in a x64 configuration), and two banks of QDR2 SRAM (32Mbits total in two x32 dual-ported banks). The FPGA uses the memories for data buffering and computational workspace.

Table 7. X6 Computing Core Devices

Feature	Device	Part Number
Application Logic FPGA	Xilinx Virtex-6 SX315T Xilinx Virtex-6 SX475T Xilinx Virtex-6 LX240T	XC6VSX315T-1FFG1156C (some cards use -2) XC6VSX475T-1FFG1156C (some cards use -2) XC6VLX240T-1FFG1156C (some cards use -2)
Memory	LPDDR2 DRAM	4x Micron MT42L256M32D4KP-25 IT

The X6 FPGA computing core connects the IO, peripherals, host communications and support features. Each IO device on the X6 modules directly connects to the application FPGA, providing tight coupling for high performance real-time IO. The FPGA logic implements an interface to each device that connects them to the controls and data communications features on

the module. Support features, such as sample triggering and data analysis, are implemented in the logic to provide precise real-time control over the data acquisition process.

The X6 module architecture is really defined by the features in the logic that connect the IO devices to data plane and system interfaces. The data plane connects the IO and computing elements using a high speed packet network. Data from IO devices such as A/Ds is made into data packets conforming to ANSI/VITA 59, specification, and enters the data plane for routing to other devices or logic elements. This packet architecture provides the X6 modules with a flexible and extensible data architecture that grows to meet application requirements.

Data transmissions to or from the host PCIe interface use the Velocia packet system to communicate with the host. The Velocia packet system implements an automated DMA transfer with the host sytem over the PCIe interface. The Velocia packet system provides both sustained high transfer rates driven by hardware and the ability to send multiple simultaneous data streams across the interface as virtual channels. Velocia packets are bundles of VITA packets containing data for the IO and signal processing.

The packet stream to the PCIe host is enqueued in the VFIFO data buffer, and subsequently packetized into Velocia packet for transfer. Packets to output devices travel in the opposite direction – from the link to the de-framer and into the VFIFO data buffer. The output IO, such as a DAC, then consumes the VITA data packets from the queue as required and dissembles them into a data stream for the output device.

The Alert Log monitors error conditions and important events for management of the data acquisition process. Alerts are used to inform the system when an important event, such as a trigger, or error has occurred. When a alert signaled, a packet is created to the system that includes vital data messages or other status information. The system receives an interrupt and can use the alert information to manage the card in its application.

The host interacts with the X6 computing core using the packet system for high speed data and over the command channel. The packet system is the main data channel to the card and delivers the high performance, real-time data capability of moving data to and from the module. Since it uses an efficient DMA system, it is very efficient at moving data which leaves the host system unburdened by the data flow. The command channel provides the PCIe host direct access to the computing core logic for status, control and initialization. Since it is outside the packet system, it is less complex to use and provides unimpeded access to the logic.

The application FPGA image is loaded at power up from onboard flash EEPROM storage.

Adding New Features to the FPGA

The functionality of the computing core can be modified using the FrameWork Logic tools for the X6 module family. The tools support development in either VHDL or MATLAB. Signal processing, data analysis and unique functions can be added to the X6 modules to suit application-specific requirements. See the *X6 FrameWork Logic User Guide* for each product for further information.

X6 PCI/PCI Express Interface

The X6 module family has a PCI and PCI Express interface for host communications. Only one interface, either PCI or PCI Express is active at one time. This allows the module to be used in both XMC IO sites and older PMC PCI bus sites. PCI Express offers transfer rates up to 8 GB/s, while PCI transfers are 256MB/s maximum.

The standard product ships with a x8, Gen1 PCIE Express interface. IP cores for x1, x4 Gen1 and Gen2 are also avialble.

NOTE: The higher speed x8 Gen PCIe core requires a -2 speed grade FPGA that must be specified at order.

PCI Express Interface

The PCI Express interface differs from the PCI interface in both implementation and performance. The PCI Express interface directly communicates from the host bus to the FPGA using high speed serial lanes (GTX). Each lane consists of one transmit and one receive wire so that the interface is full duplex. The lanes can be used in x1, x4 or x8 configurations as negotiated by the host system at boot up as part of the port initialization. The interface always seeks the highest performance first, starting with x8 lanes at 2.5 Gbps (Gen1) or 5GBs for Gen2. If the host machine cannot support this interface, the highest possible rate is negotiated, degenerating to a x1 2.5 Gbps interface at worst.

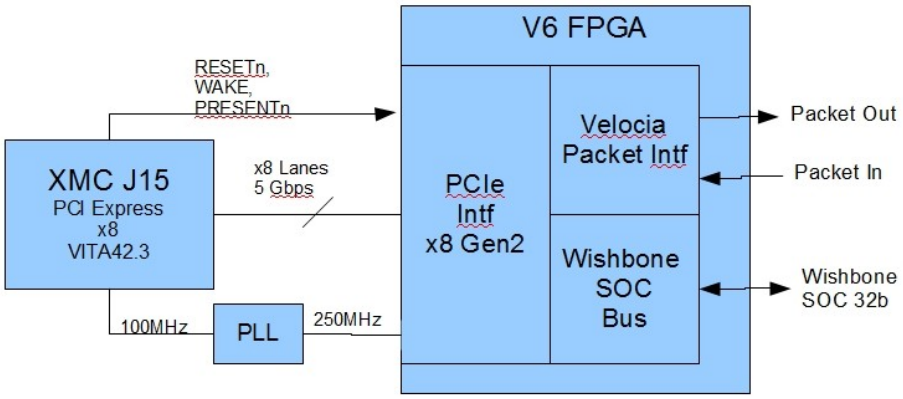


Figure 16. PCI Express Interface Block Diagram

PCI Interface

The PCI bus is a 32-bit, 66MHz interface compatible with most PMC sites. The X6 uses a bridge from the PCI bus to the FPGA that translates the PCI bus to a single lane (x1) PCI Express interface. The bridge device (Pericom PI7C9X111SLBFDE) is non-intelligent and simply provides connectivity to the FPGA. In this way, the PCI and PCI Express interfaces are symmetric to the FPGA design, both appearing as PCI Express to the designer.

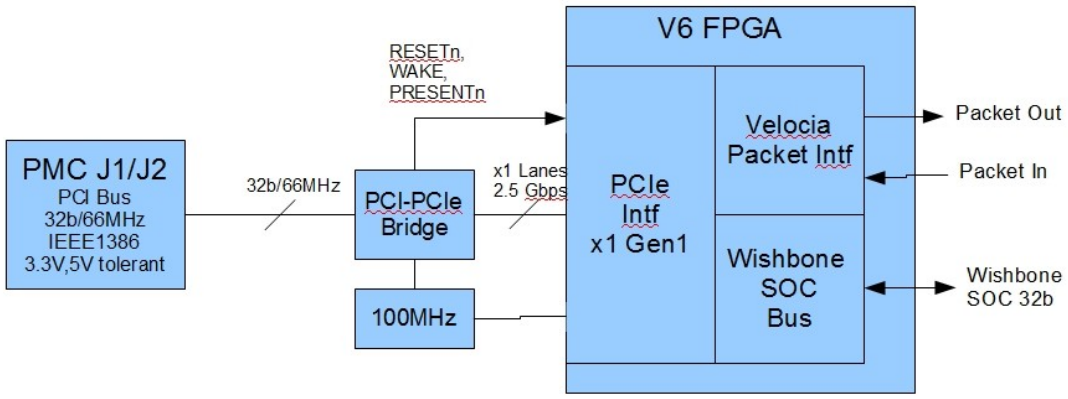


Figure 17. PCI Interface Block Diagram

Configuring The Host Interface

Host cards are usually either PCI Express or PCI, not both. For performance, the PCI Express interface should be used. The PCI interface is used primarily for compatibility to systems without PCI Express. It is normal that some of the connectors will not have mates when the host is PCI Express only and is not harmful.

The choice of either PCI or PCI Express interface requires that a different logic image be used in each type. This logic defines the interface type and must match the desired interface.

Host Interface	Logic Image	Notes
PCI Express x8 Gen1	x5_400m.bit	Max data rate is 8GB/s, Gen1 . Sustained rates ~1GB/s.
PCI	x5_400m_x1_bridge.bit	Max data rate is 256MB/s. Sustained rates ~180MB/s.
PCI Express x8 Gen2	x5_400m_x8_gen2.bit	Max data rate is 8GB/s, Gen2 . Sustained rates ~2GB/s.

Table 8. Logic Images for PCI and PCI Express Hosts

PCI and PCI Express Standards

The following standards govern the PCI Express interface on the X6 XMC modules.

Standard	Describes	Standards Group
PCI Express 2.1	PCI Express electrical and protocol standards. 5 Gbps data rate per lane per direction.	PCI SIG (http://www.picmg.com)
PCI Local Bus 3.0	PCI Local Bus, a 32-bit bus operating at up to 66 MHz.	PCI SIG (http://www.picmg.com)
ANSI/VITA 42	XMC module mechanicals and connectors	VITA (www.vita.org)
ANSI/VITA 42.3	XMC module with PCI Express Interface.	VITA (www.vita.org)

Table 9. PCI/PCI Express Standards Compliance

FPGA Integration with the PCI/PCIe Host Data Path

A data plane and control plane are implemented in the X6 family that use the PCI Express interface to communicate with the host. The data plane uses the Velocia packet DMA channel and the control plane uses a Wishbone SOC bus. The Velocia DMA channel automates the data transfers using a packet system, achieving data rates up to 2000 MB/s sustained transfer rates (requires x8 Gen2 PCIe core). This high speed path is complemented by the Wishbone SOC bus for system command, control and status. The Wishbone SOC provides a flexible system bus interconnecting the logic components.

Application Logic Interface	Max Data Rate	Typical Use
Velocia Packet Interface	2000 MB/s sustained to host memory. (x8 Gen2 PCIe core)	Velocia packet system interface - main path for data communications
Wishbone SOC Bus	20 MB/s sustained	System-On-Chip bus for command, control and status. This bus is used to connect logic components in the FPGA.

Table 10. Interfaces from PCI Express to Application Logic

Additional information is available in the *X6 Framework Logic User Guide* for each X6 module.

Data Buffering and Memory Use

There are four banks of memory attached to the application FPGA useful for data buffering and computational memory. All of the memory banks are LPDDR2 memory, each with an independent connection to the FPGA. Logic components in the Framework Logic implement memory controllers, used for functions such as data queues, pattern generators, and computation buffers.

DRAM Devices

The DRAM devices are LPDDR2 memories. These devices are similar to DDR2, with improvements resulting in lower power consumption and more flexibility for migration to larger memories. The devices can operate at speeds up to 533 MHz.

Memory	Size	Device
U23, U25, U26, U27	1GB; 256M x 32bit x 4banks	Micron MT42L256M32D4KP-25 IT

Table 11. X6 DRAM Devices

DRAM Connections to FPGA
Each DRAM device has a Command-Address bus and a data bus. These buses connect directly to the FPGA, where they are controlled by a memory interface component. This logic component provides the interface protocol and timing required to communicate with the DRAM. This includes initialization, data transaction controls, memory refresh, and power controls.

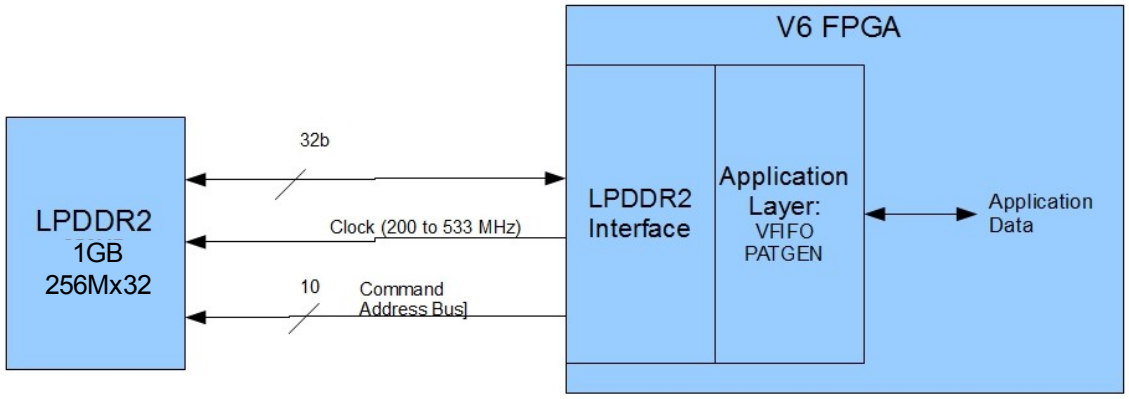


Figure 18. X6 DRAM Interface

Data Buffer DRAM
Logic in the FPGA implements a data buffer, or "Virtual FIFO" (VFIFO), for the system interface. Data packets transacted with the host computer are en-queued in the VFIFO, using the DRAM as the memory for this queue. Application logic can use the DRAM interface for any type of data, for buffering or computation memory, through the LPDDR2 interface. The Framework Logic implements a data buffer with one or more queues for the A/D and D/A streams as appropriate for the particular X6 module.

The X6-TX has special support for pattern generation using the DRAM memory bank. The pattern generation mode provides dynamic pattern generation using DRAM memory bank that is driven by control and data packets from the host. This allows the DRAM to be used as an arbitrary waveform generator for applications such as communications testing, RADAR stimulus generation and ultrasonic stimulus.

The buffer memory data transfer rates are limited by the maximum clock rate for the memory device and the speed of the logic device. Each X6 module has a memory clock rate selected so that it supports the IO requirements of the card. IN most

cases, this means that the clock rate is lower than the maximum clock rate that the DRAM can support, simply because the power consumption is lowered and it is easier to meet timing during design.

X6 Module	Maximum DRAM Clock Rate (MHz)	Maximum Memory Bandwidth (MB/s)	FrameWork Memory Clock (MHz)	FrameWork Logic Memory Bandwidth (MB/s)
X6-RX	533	4264	333	2664
X6-400M	533	4264	333	2664

Table 12. Buffer Memory Bandwidth

Serial FLASH Interface

FLASH Memory Description
A serial FLASH memory on the X6 modules is used for a variety of purposes including program memory for embedded processors, module information and calibration data. For X6 modules, one sector of the FLASH is dedicated to module information and calibration data, consuming 1/64 of the memory. Sectors may be protected in the device for data security.

Memory	Size	Device
U32	32MB	Microchip SST25VF032B-80-4I-S2AF

Table 13. X6 Serial FLASH Device

The interface to the serial FLASH is an SPI bus that is controlled by the PCI logic device. The SPI bus is relatively slow, about 10Mb/s, so the data is usually read out of the FLASH at initialization time by the application software and written into memory or registers in the application logic for real-time use. Calibration values for the analog are an example of this use, where the calibration coefficients are read from the FLASH then written into the error correction logic for real-time data correction.

The FLASH has a write cycle limit of 100K cycles, so it should only be written to when calibration is performed or configuration information changes. Once the write cycle duration limit is exceeded, the device will not reliably store data any more.

Caution: As delivered from the factory, the serial FLASH contains the calibration coefficients for the analog and is pre-programmed at factory test. Do not erase these coefficients or calibration will be lost.

Use the baseboard `IdRom()` method to obtain a reference to the internally-managed `IUsesPmcEeprom` object, as shown below:

```
// Open the module
Innovative::X6_RX Module;
Module.Target(0);
Module.Open();
```

```
// Create a 50-32-bit-word section at offset zero in ROM user space
PmcIdromSection Section1(Module.IdRom().Rom(), PmcIdrom::waUser, 0, 50);
// Create a 50-32-bit-word section at offset 50 in ROM user space
PmcIdromSection Section2(Module.IdRom().Rom(), PmcIdrom::waUser, 50, 50);

// Write to ROM
for (int i = 0; i < 50; ++i)
       Section1.AsInt(i, i*2);
Section1.StoreToRom();

for (int i = 50; i < 100; ++i)
       Section2.AsFloat(i, static_cast<float>(i*2));
Section2.StoreToRom();

// Read from ROM
Section1.LoadFromRom();
for (int i = 0; i < 50; ++i)
       int x = Section1.AsInt(i);

Section2.LoadFromRom();
for (int i = 50; i < 100; ++i)
       float x = Section2.AsFloat(i);
```

Digital I/O

The X6 series has a digital I/O port accessible using the P16 or JN4 connector. These bits are direct connections to the FPGA and are used in many custom applications for controls, communications and status signals. The digital IO bits use either LVDS or LVCMOS2.5 IO standards. LVDS signaling is recommended for high speed and improved noise immunity.

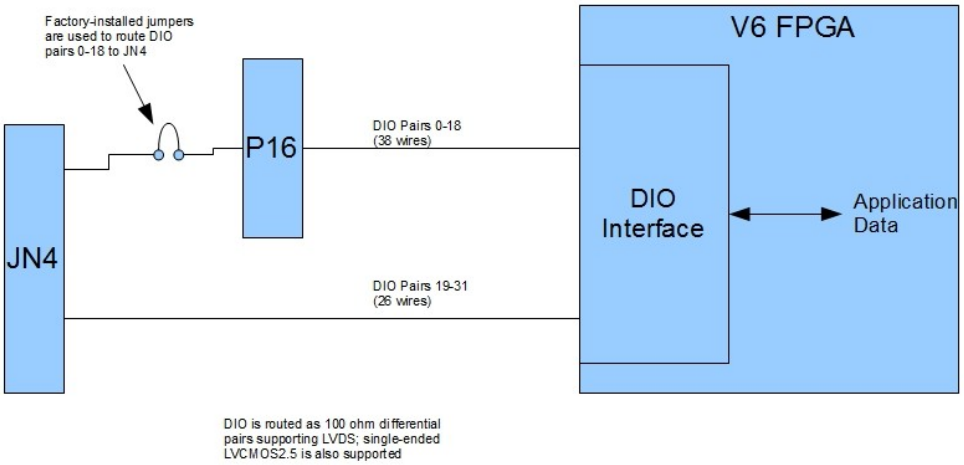

Factory-installed jumpers are used to route DIO pairs 0-18 to JN4
P16
JN4
DIO Pairs 0-18 (38 wires)
DIO Pairs 19-31 (26 wires)
V6 FPGA
DIO Interface
Application Data
DIO is routed as 100 ohm differential pairs supporting LVDS; single-ended LVCMOS2.5 is also supported

Figure 19. X6 Digital IO Block Diagram

Digital IO Connections

The digital bits are configured at the factory to use either to P16 or JN4. The default configuration is to use JN4 for the digital IO connections. Pinouts for each connector or shown in the hardware section of this manual.

DIO Pairs	Connctor	Notes
0-18	JN4 (factory default) P16 (optional configuration)	All pairs are connected to JN4 as a factory default. Optional configuration removes jumpers for P16.
19-31	JN4	

Table 14. DIO Connector Configurations

Digital IO Electrical Characteristics

The digital IO bits are usable as either LVDS or LVCMOS 2.5. This IO standard is chosen in the logic design. The FrameWork Logic for X6 uses LVDS for all digital IO connections.

Caution! Voltages above 2.6V will damage the Virtex-6 device. Do NOT connect 3.3V or 5V logic directly to the digital pins as damage may occur.

LVCMOS 2.5V Pins

Warning: the LVCMOS DIO pins are **NOT** 3.3 or 5V compatible. Input voltage must not exceed 2.8V during normal operation. Undershoot and overshoot must be limited: see the Xilinx Virtex-6 User Guide for details.

Parameter	Value	Notes
Input Voltage	Max = 2.8V Min = -0.3V	Exceeding these will damage the FPGA
Output Voltage	'1' > 2.1V '0' < 0.4V	
Output Current	+/-12mA	FPGA can be reconfigured for custom designs for other drive currents.
Input Logic Thresholds	'1' >= 1.7 VDC '0' < 0.7 VDC	
Input Impedance	>1M ohm || 15 pF	Excludes cabling

Table 15. LVCMOS 2.5 Digital IO Bits Electrical Characteristics

LVDS 2.5 Pins

The LVDS signal pairs support high speed signaling. All pairs are routed as 100 ohm differential impedance, 50 ohm single-ended. All LVDS pairs must be terminated to 100 ohms. For inputs, designers should instantiate on-chip termination in the Virtex-6 logic. For outputs, the signal destination must have 100 ohm termination impedance.

Warning: the LVDS 2.5 DIO pins are **NOT** 3.3V or 5V compatible. Input voltage must not exceed 2.8V during normal operation. Undershoot and overshoot must be limited: see the Xilinx Virtex-6 User Guide for details.

Parameter	Value	Notes
Input common-mode voltage	1.2V typical 0.3V min; 2.2V max	For differential input voltage = +/- 350 mV
Input Differential Voltage	1000mV max 100mV min	100 ohm differential load
Output High Voltage P or N	1.765V max	100 ohm differential load
Output Low Voltage P or N	0.715V max	100 ohm differential load
Output Differential Voltage	840mV max 350mV min	100 ohm differential load

Table 16. LVDS2.5 Digital IO Bits Electrical Characteristics

Software Support

The Framework Logic configures these bit I/O connections as a simple port that has programmable bit direction and data by the host. The port is configured and accessed directly from the PCI Express host.

The digital I/O hardware is controlled by the `IUsesDioPort` class. Its properties:

Table 17. IUsesDioPort Class Operations

Function	Type	Description
DioPortConfig()	Property	Configures banks of bits for input or output
DioPortData()	Property	Broadside Read/Write to low-order 32-bits of DIO.

Typical use of the digital IO port involves first configuring the port using the Config() operator. This sets the byte direction and the clock mode. The port is then ready for read/write configurations to each port. For instance:

```
// Open the module
```

```
Innovative::X6_RX Module;
Module.Target(0);
Module.Open();

// All bits input
Module.Config(0x0);
// Read the state of the port
volatile short x = Module.DioPortData();

// All bits output
Module.Config(0x3);
// Toggle the state of all output bits
while (1)
    Module.DioPortData(~Module.DioPortData());
```

FrameWork Logic Digital IO Implementation

Digital I/O port activity is controlled by the digital I/O data register in the Framework Logic. The digital IO can be used for simple bit IO in this mode. Data written to bits 31..16 of the register are output, data read from the register in bits 0-15 are the inputs.

Pairs	IO Standard	Function
15..0	LVDS	Input
31..16	LVDS	Output

Figure 20. Default Digital IO Configuration

Notes on Digital IO Use

The digital I/O on X6 modules, as supported using the standard FrameWork Logic, is intended for low speed bit I/O controls and status. Application logic can implement much higher speed and sophisticated interfaces by modifying the interface to the pins to construct higher speed data transfers, controls and timing signals as required by the application.

The X6 FrameWork Logic user Guide details logic supporting the digital IO port and gives the pin information for customization.

P16 Communication Ports

The X6 series implement a high speed communication ports on the XMC P16 connector used as a secondary data path on the module,. These ports are Aurora protocol for the Framework Logic. The high speed serial ports are connections to the FPGA GTX ports and can be used for other protocols or custom implementations.

The P16 connections on the X6 are compatible with the VITA 42.0 secondary connector specification, and provide eight transmit and receive pairs implemented using Virtex-6 GTX links. A programmable clock reference is provided on board for the GTX links.

The P16 connector pinout is shown in the hardware chapter.

GTX Connections to P16

Pinouts for the P16 connector showing the transmit and receive pair locations are given in the Connectors section. The following table gives the Rocket I/O pin allocations on the Virtex-6 that connect to each of the P16 signals.

Link	Lane	P16 Pins	P16 Signal	Virtex-6 FG1156 Pin Number	Virtex-6 GTX Signal Identifier
0	0	A1/B1	TXP0/N0	AP1/AP2	GTXTXP0_112/GTXTXN0
0	0	A11/B11	RXP0/N0	AP5/AP6	GTXRXP0_112/GTXRXN0
0	1	D1/E1	TXP1/N1	AN3/AN4	GTXTXP1_112/GTXTXN1
0	1	D11/E11	RXP1/N1	AM5/AM6	GTXRXP1_112/GTXRXN1
0	2	A3/B3	TXP2/N2	AM1/AM2	GTXTXP2_112/GTXTXN2
0	2	A13/B13	RXP2/N2	AL3/AL4	GTXRXP2_112/GTXRXN2
0	3	C3/D3	TXP3/N3	AK1/AK2	GTXTXP3_112/GTXTXN3
0	3	D13/E13	RXP3/N3	AJ3/AJ4	GTXRXP3_112/GTXRXN3
1	0	A5/B5	TXP4/N0	AH1/AH2	GTXTXP0_113/GTXTXN0
1	0	A15/B15	RXP4/N0	AG3/AG4	GTXRXP0_113/GTXRXN0
1	1	D5/E5	TXP5/N1	AF1/AF2	GTXTXP1_113/GTXTXN1
1	1	D15/E15	RXP5/N1	AF5/AF6	GTXRXP1_113/GTXRXN1
1	2	A7/B7	TXP6/N2	AD1/AD2	GTXTXP2_113/GTXTXN2
1	2	A17/B17	RXP6/N2	AE3/AE4	GTXRXP2_113/GTXRXN2
1	3	C7/D7	TXP7/N3	AB1/AB2	GTXTXP3_113/GTXTXN3
1	3	D17/E17	RXP7/N3	AC3/AC4	GTXRXP3_113/GTXRXN3

Table 18. P16 GTX Assignments and Use

GTX Electrical Characteristics

The GTX lanes are high speed serial port connections to the Virtex-6. These are specialized serial ports intended for use as communications ports and include many advanced features for data transmission and reception at speeds up to 5 Gbps. They are generally not useful for any other purpose.

The GTX lanes must be AC-couple at the transmitter end. Receive (RX) lanes to the X6 must be AC-coupled at the transmitter. Transmit lanes are AC-coupled on the X6. IO standard is CML.

Parameter	Value	Notes
Differential pk-to-pk Input Voltage	175 mV min 2000 mV max	AC- coupled externally.
Absolute Input Voltage	-400 mV min	DC-coupled.
Common Mode Input Voltage	800mV	Typical
Differential pk-to-pk Output Voltage	1000 mV max	Programmed to maximum swing output.
Common Mode Output Voltage	800mV typical 1000mV max	
Lock Time	1 ms	Max
Input common-mode voltage	1.2V typical 0.3V min; 2.2V max	For differential input voltage = +/- 350 mV
Input Differential Voltage	2000mV max 800mV typ 210mV min	100 ohm differential termination
Differential Resistance	90 to 120 ohms	100 ohms nominal; input and output

Table 19. GTX Signal Electrical Characteristics

GTX Reference Clocks

The X6 ports on P16 have an on-card programmable reference clock PLL. This PLL is capable of generating a wide range of frequencies to support many standards. The reference for the PLL a 10 MHz, 1ppm (0.28 ppm optional) reference clock. This very stable reference allows the GTX to meet stability requirements for many communications standards such as Serial Rapid IO, Ethernet and SONET.

Parameter		Units	Notes
Frequency Range	0.16 to 350	MHz	LVDS
Tuning Resolution	1	ppm	
Stability	1	ppm	Using on-card 10MHz reference 1ppm; 0.28 ppm option is available
Jitter	0.7	ps RMS	
Lock Time	25	ms	Max

Table 20. GTX PLL Reference Clock Characteristics

The GTX can also receive a clock from P16 if the logic is recompiled to use the alternate GTX clock input. This clock input directly connects to the GTXs. The IO standard is LVDS. For most applications, this clock is in the range of 125 to 250 MHz.

Parameter	Value	Notes
Frequency Range	62.5 to 650 MHz	
Rise/Fall Time	200 ps	Typical
Duty Cycle	45-55%	
Lock Time	1 ms	Max
Input common-mode voltage	1.2V typical 0.3V min; 2.2V max	For differential input voltage = +/- 350 mV
Input Differential Voltage	1000mV max 100mV min	100 ohm differential termiantion
Terminattion	100 ohms	nominal

Table 21. GTX Alternate Reference Clock Input Characteristics

Aurora Ports

The Framework Logic implements dual Aurora communication cores for the secondary ports on P16. The two Aurora links are 4 lanes each, providing up to 2GB/s maximum transfer rates (5 Gbps operation, full duplex). The ports are independent in the FrameWork Logic, but can be paired into a single 8 lane port in the logic design.

The Aurora ports support multiple virtual channels across the link, enabling both high speed data transfer as well as low speed command, control and status functions between cards. The slower speed channel is referred to as the “command channel”. For the designer, this structure allows the ports to be used primarily as a fast data channel to another card while the command channel provides a means for the control and status words used for flow control, system coordination and initialization.

Customization for the Aurora ports and the data format is described in the *X6 FrameWork Logic User Guide* for each module.

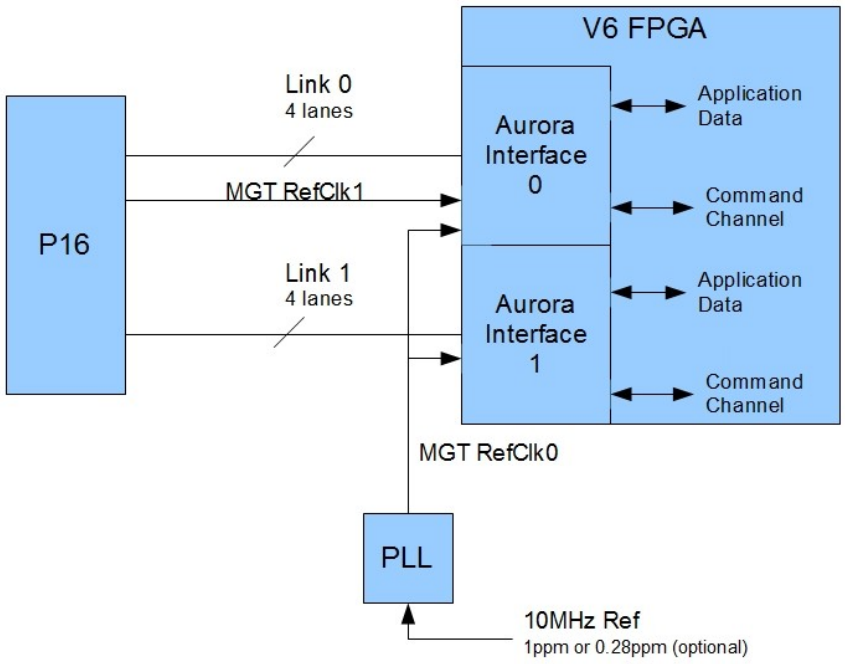


Figure 21. X6 Aurora Ports on P16

Connecting Multiple X6 Modules Using Aurora Ports

The Aurora ports are integrated into the data plane for the logic, thereby allowing ports to communicate with system cards. VITA data packets can flow to or from the Aurora cores to components in the FPGA. Data is routed as VITA packets to system elements, enabling larger channel count systems, distributed processing, and coordinated system functions.

Multiple cards can share data in a system over the Aurora ports using either Innovative adapter cards supporting Aurora cabling, or in systems supporting an Aurora backplane. The example cluster of three X6 modules shows a ring topology implemented using the Aurora ports. The modules are mounted to x8 PCIe Adapter (P/N 80173 or P/N 80259) and plugged into a rackmount or desktop PCIe system. Each adapter card pins out the ports on SATA connectors. The cards are connected using SATA cables that connect the transmit port of each card to a the receiver on another card. Each cable carries two lanes, so two cables are required per link (the Aurora port is x4 lanes in each direction). Other topologies can be easily implemented by rearranging the cables.

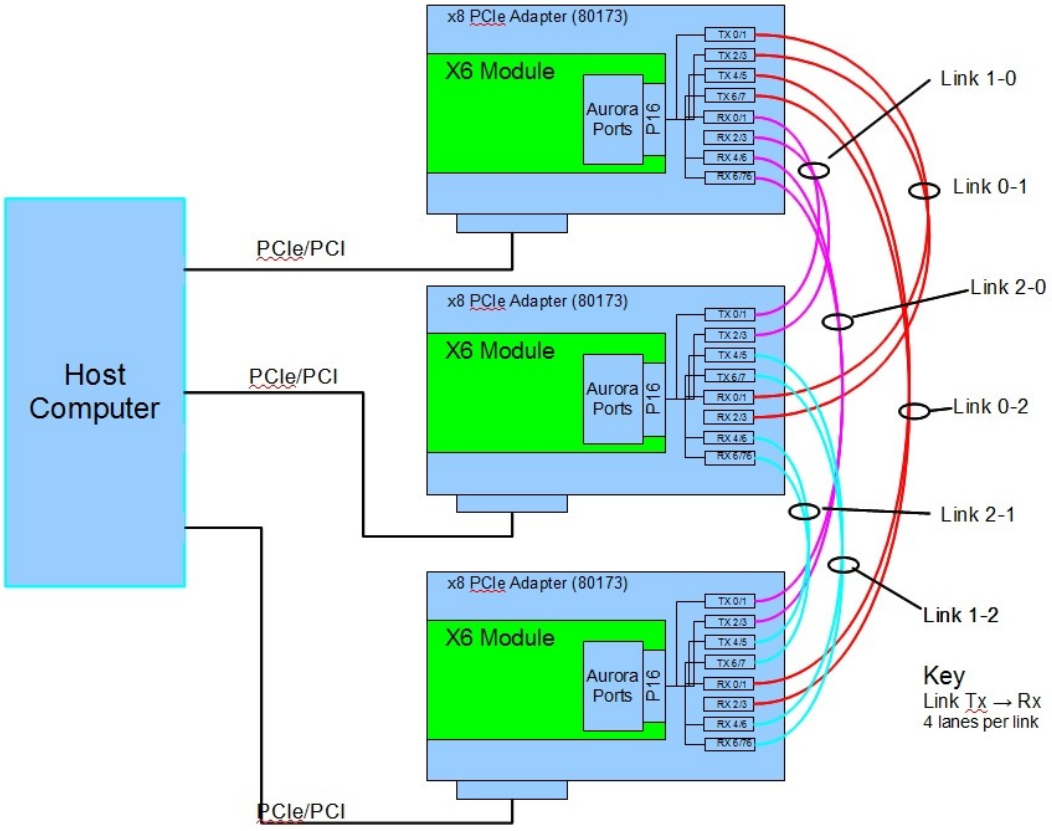


Figure 22. X6 Module Cluster Using Aurora

The modules can communicate at speeds up to 900 MB/s full-duplex in this topology using 2.5 Gbps lanes. The cables limit the lanes to a maximum of 3.125 Gbps in this configuration.

System Timing Features on P16

System integration features include timing signals on P16 for system synchronization and coordination. These features allow the use of system-wide clocks, triggers and timing events to generate local sampling and control signals.

Signal	P16 Pin	Description	Notes
H_PPS	D19	Pulse-per-Second timing input. This signal is often from GPS or other timing source.	LVCMOS2.5 IO standard.
H_TRIG0	A19	Trigger input 0 used for system coordination.	LVCMOS2.5 IO standard.
H_TRIG1	B19	Trigger input 0 used for system coordination.	LVCMOS2.5 IO standard.
FMC_CLKP/N	A9/B9	Sample clock input. This signal can be used as a sample clock or PLL reference.	LVDS 2.5 IO standard.
FMC_VADJ	E19	Reference voltage input for the JN4/P16 digital IO. This reference voltage allows the DIO to be used with different IO standards including 2.5V, 1.8V, and 1.5V.	This voltage must never exceed 2.5V. Current requirement is <100 mA.

Table 22. System Integration Signals on P16 Connector

The H_PPS and FMC_CLKP/N signals can be used to integrate the X6 module with system timing signals from a GPS or other time reference. Many timing references such as GPS, IEEE 1588 LAN timing, and ARINC include both a PPS signal and reference clock. The PPS is generally used as an arming signal for system timing, while the reference clock is used to generate synchronous sampling clocks.

Thermal Protection and Monitoring

The X6 includes a temperature monitor for the Virtex6 device. This monitor connects directly to the FPGA temperature diode connections so that the FPGA die temperature is directly measured. The temperature monitor does not use the FPGA logic or system monitor for this measurement so that it can protect the FPGA in case of failure.

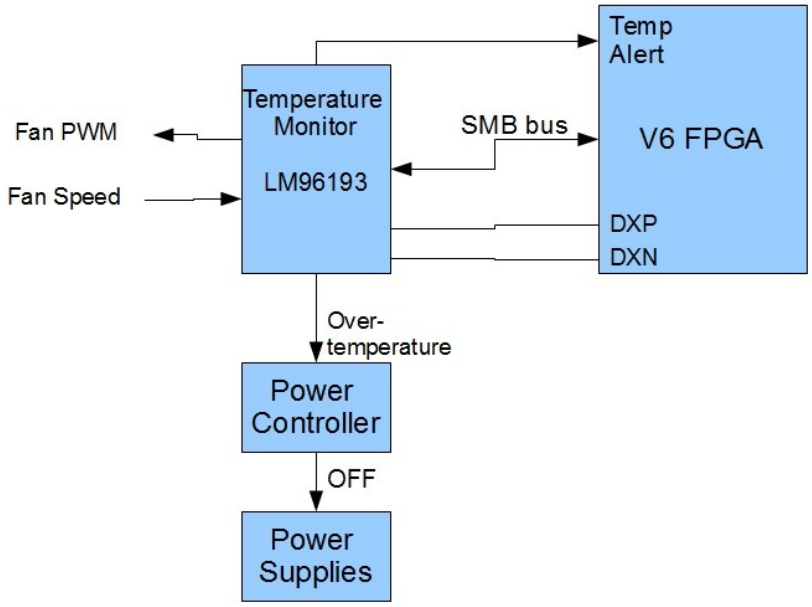


Figure 23. X6 Temperature Monitoring

The monitor watches the FPGA die temperature and compares it to a critical and alert threshold. These thresholds are programmed into the temperature monitor device to override the defaults. Software functions in Malibu software are provided for programming this device.

Module Environment Rating	Rated Operating Temperature (C)	Programmed Alert Threshold (C)	Programmed Critical Threshold (C)	Default Alert Threshold (C)	Default Critical Threshold (C)	Maximum Allowable (C)	Notes
L0	0 to +50C	50	85	85	110	85	Commercial temperature device
L1	-40 to +85	65	100	85	110	100	Industrial temperature FPGA
L2	-20 to +65	65	100	85	110	100	Industrial temperature FPGA
L3	-40 to +70	65	100	85	110	100	Industrial temperature FPGA
L4	-40 to +85	65	100	85	110	100	Industrial temperature FPGA

Table 23. X6 Temperature Threshold and Limits

When the alert threshold is e exceeded, the monitor alert signal to the FPGA is fired. This temperature alert is passed to the system through the packet system. Host software is therefore notified of the temperature alert and can take appropriate action. When the critical threshold is exceeded, the temperature module disables power. In the event of an over-temperature condition, the logic, memory interface, and analog power supplies are disabled, shutting down power to most of the X6 module. The host system must re-enable power to start the module again.

In the event that the temperature threshold must be ignored, the software can reprogram temperature monitor to set the threshold to maximum (255C).

Temperature Readout

The temperature sensor on the card monitors the FPGA die temperature. This gives a good indication of the card temperature since the FPGA is usually the hottest device and is located in the center of the card.

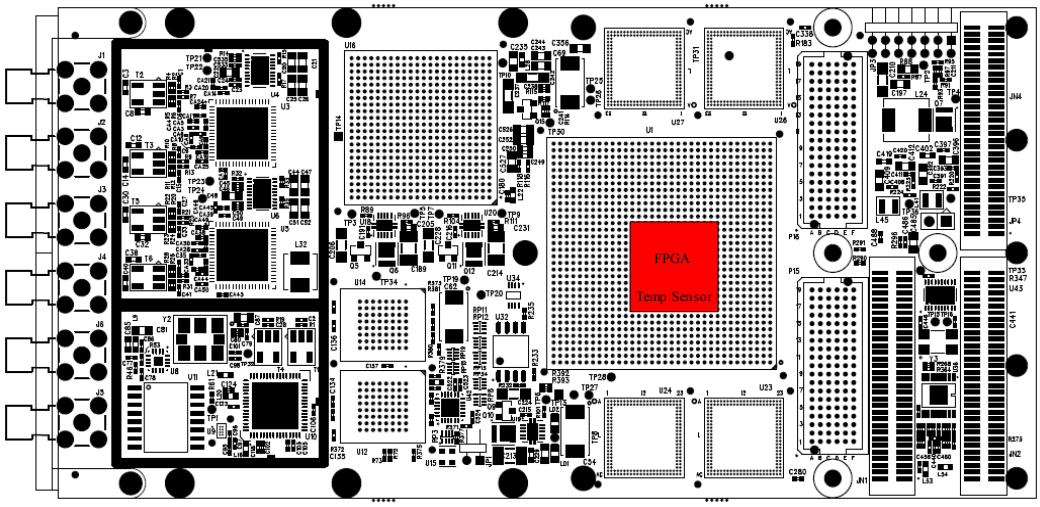


Figure 24. Temperature Sensor Location

Temp Sensor	Range	Accuracy	Monitor	IO Bus	Software Functions
FPGA Die	-125C to +125C	+/-1.5C	FPGA on die transistor junction	SMB bus via LM96193 monitor	Temperature readback and thresholds

Table 24. X6 Temperature Sensor

Software Support for Temperature Monitoring

Software support tools provide convenient access to the temperature and thermal controls. These should be used in application programming configure and monitor the temperature, as illustrated below:

```
// Open the module
Innovative::X6_RX Module;
Module.Target(0);
Module.Open();

// Create reference to thermal management object on module
const LogicTemperatureIntf & Temp(Module.Thermal());

// Read current temperature
float t = Module.LogicTemperature();

// Read/write current warning temperature
float t = Module.LogicWarningTemperature();
Module.LogicWarningTemperature(70.0);

// Read current failure temperature
float t = Module.LogicFailureTemperature();

// See if the module is in thermal shutdown
bool state = Module.Failed();
```

System Thermal Design

The X6-RX can dissipate upwards of 25 watts depending on the features in use and details of the logic design, such as the rate of data processing. Forced air cooling may be required depending on the power dissipation and the ambient operating temperatures. This requirement is highly application dependent and must be evaluated for each application and installation.

Fan Control

The X6 temperature monitor integrates a fan control circuit with the temperature sensor. This fan control has a Pulse-Width Modulated (PWM) output to control fan speed and tachometer input from the fan. The fan control signals are on J15 connector (see hardware chapter for pinout). The LM96193 must be programmed to set the PWM resolution and tachometer scale factor, which is unique for most fans. This programming is done over the SMB bus connection to the Virtex-6 FPGA. Consult the National Semiconductor LM96193 documentation for details.

The fan PWM output is NOT capable of driving most fans. A separate power transistor must be used with the PWM signal as the control. A 1K pullup on the PWM signal makes the fan ON by default if the LM96193 is not programmed.

The fan tachometer signal is a direct input to the LM96193. The LM96193 must be programmed for the tachometer input to be active.

An example fan control circuit is shown in the following figure. The fan connects to the circuit through the 3-pin header since it is usually mounted to the chassis. A power MOSFET (Fairchild Semi NDS355N) is used for the fan motor control. The FAN_CTL# signal is the PWM from the LM96193 and is OR'd with the XMC_PRESENT# signal. When both are true, the gate to the MOSFET is ON. This particular MOSFET has an on threshold of 2V and a current capacity (Id) of 1.7A. This allows the power MOSFET to turn on with a 3.3V logic signal and control a moderate size fan. Check your fan specification for current requirement. The fan tachometer feedback to the LM96193 is pulled up to 3.3V on the module using a 1K ohm resistor.

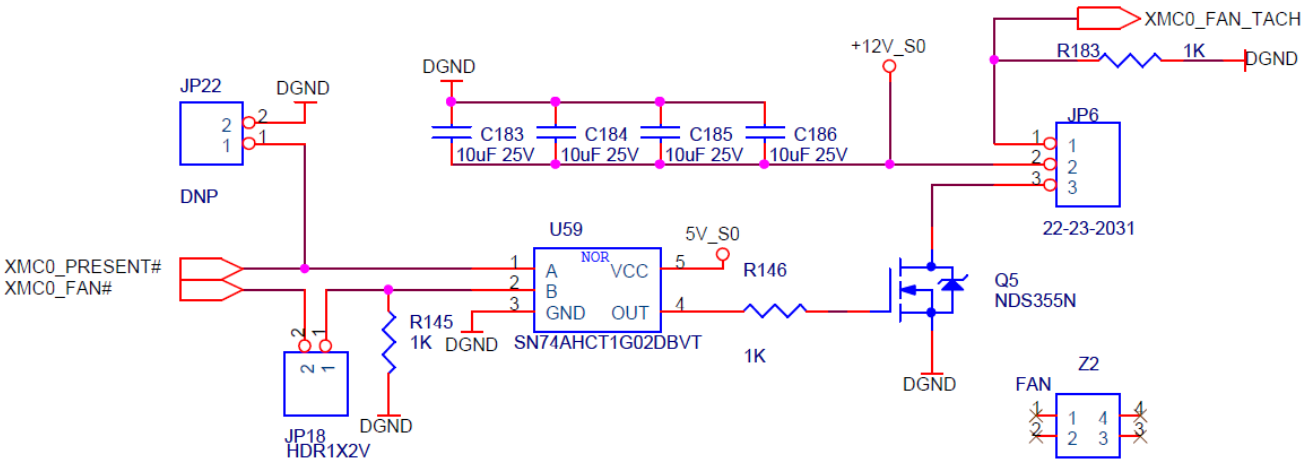


Figure 25. Example Fan Control Circuit

Conduction Cooling

If forced air cooling is not used, conduction cooling is another method of dissipating the module heat. A thermal plane in the card is attached to thermal conduction surfaces on each side of the module. The thermal plane has NO electrical connection in the module and cannot be used as a ground.

The card can then be cooled by mounting the card on host card that supports conduction cooling per VITA specification 20. The conduction cooling features are a subset of the VITA 20 specification. The conduction cooling method allows the module heat to be conducted to the chassis.

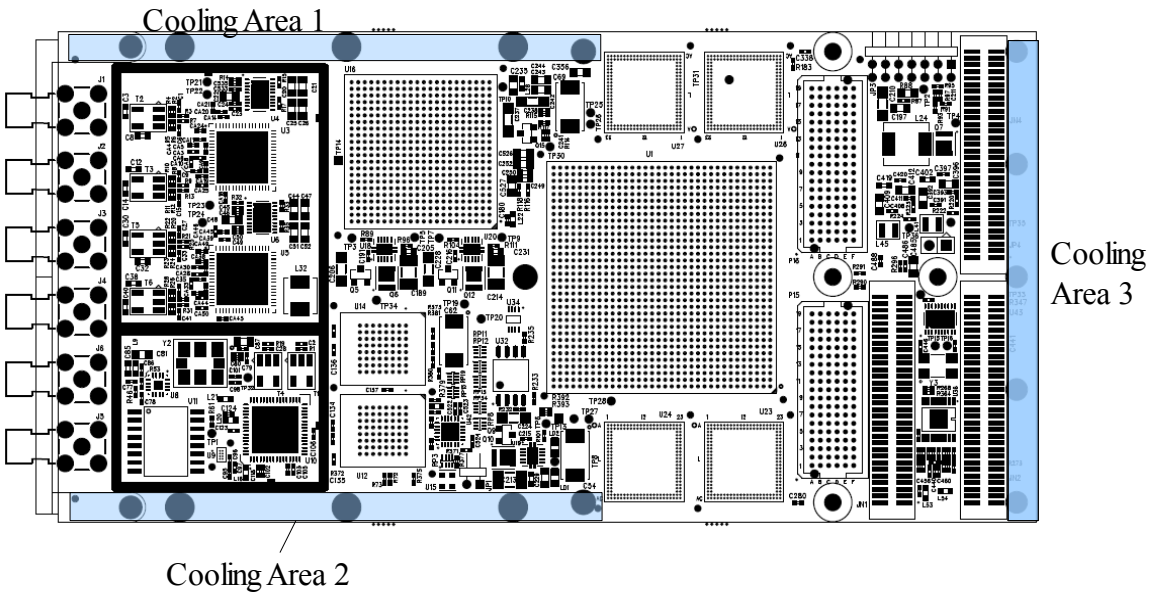


Figure 26. X6 Conduction Cooling Thermal Connection Areas

Hardware	Description	II P/N	Notes
Rear cooling bar	Aluminum bar with M2 screw holes, 10mm mounting height, for X3/X6 modules	61122	For cooling area 3
Side cooling bar	Aluminum bar with M2.5 screw holes, 10mm mounting height, for X5/X6 modules.	61121	For cooling areas 1 & 2
Screws	M2x5, pan head	63167	10 total per module
Screws	M2.5x5, pan head	63130	12 total per module

Table 25. X6 Conduction Cooling Hardware

X6 FPGA Heat Spreader

The X6 cards have a heat spreader for the FPGA that spreads the heat across a large surface area and also conducts heat to the thermal plane. This heat spreader helps to conduct heat from the FPGA to the thermal plane, thereby reducing the heat concentration in the FPGA.

The heat spreader mounts to the card with four M2 screws. A thermally conductive pad on the FPGA top surface provides a conduction path to the heat spreader.

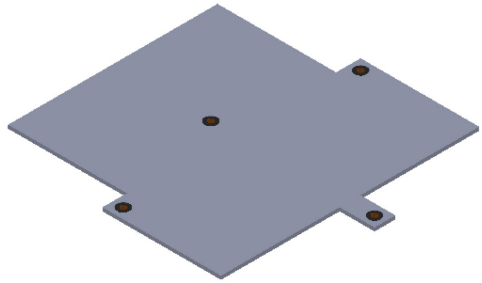


Figure 27. X6 Heat Spreader for FPGA.

Caution! If the heat spreader is removed for any reason, extreme caution should be observed in its removal. The thermal pad has adhesive to both the spreader and FPGA that makes it difficult to remove. It is best to mildly heat (<80C) the entire assembly after the screws are removed, then twist the heat spreader gently until it can be easily removed.

X6 Power Saving Features

The X6 modules have many controls for reducing the power consumption. Many devices can be powered down if they are not used by the application, or can be turned off for a sleep mode. This can reduce average power consumption.

Feature	Power Consumption	Notes
DRAM banks (4 total)	~750 mW per bank	DRAM shutdown control turns off device. All data is lost. Power up time is approximately 1 ms.
P16 GTXs (8 lanes total)	~150 mW per lane	P16 GTX s are off by default. It is also possible to stop the GTX clock for reduced power. Lane link is lost and must be re-initialized on power up.
PCI interface	600 mW	This interface is OFF unless enabled.
Analog I/O	Varies by module. X6-RX : 3W	Most modules have analog power control. Power-up time should be observed to insure analog performance.
Analog Sample Clock	1.5W	The sample clock PLL and VCXO may be powered down. The PLL is most of this power consumption and can be powered down through the PLL control registers.

Table 26. X6 Power Control Features

Another major factor in power consumption is the FPGA clock rate and size of the FPGA design. Designers should use the lowest practical clock rate to save power. The power savings can be dramatic for large designs, with many Watts saved from one design to another. The Xilinx FPGA design tools include Power Analyzer, which gives a detailed report for each FPGA design. This tool should be used to optimize the design for lower power by identifying the portions of the design that consume the most power.

Power Supply Controls

Individual power enables are provided on X6 modules that can selectively power down portions of the card not in use. Each module has unique power supplies for its IO and special functions that must be considered so that core functions required for the module to wake up from low power mode.

Module	Power Controls	Affected Subsystem	Subsystem Power Consumption	Notes
X6-RX	1.1V	DDC ASIC	Up to 5W	
	DDR_VTT	LPDDR DRAM banks	1W	Affects all DRAM banks. No DRAM can be used if this is off.
	1.2V	LPDDR DRAM banks and DDC	Up to 2W	This power is also cascaded to the 1.1V supply so if it is off, then the 1.1V must be off also.
	GTX	All GTX s used for PCIe, PCI and P16 GTX interfaces.	2W	These supplies can only be turned off if no GTX are required, such as in standalone operation.
	A/D	A/D channels, one enable per channel pair 0/1 and 2/3	750mW per channel pair	Allow 1s warm-up time after power up for stabilization.
X6-RX	DDR_VTT	LPDDR DRAM banks	1W	
	1.2V	LPDDR DRAM banks and DDC	Up to 2W	This power is also cascaded to the 1.1V supply so if it is off, then the 1.1V must be off also.
	GTX	All GTX s used for PCIe, PCI and P16 GTX interfaces.	2W	These supplies can only be turned off if no GTX are required, such as in standalone operation.
	A/D	A/D channels, one enable per channel.	2.5W per channel. Dissipates 350 mW per channel in power-down mode. Does not include output amp for DC-coupled version.	Allow 1s warm-up time after power up for stabilization.
	D/A	D/A channels, one enable. (Both channels on or off only.)	1.4W for both channels. Reduces to ~20mW in power down. Does not include output amp for DC-coupled version.	Allow 1s warm-up time after power up for stabilization.

Table 27. X6 Power Supply Controls

These power supply control bits are accessed through writes to the CPLD control registers. Consult technical support for information on use of this feature.

Led Indicators

The X6 modules have two LEDs for use by application logic, plus a module ready LED. By default, the Framework Logic image lights the blue LED when the FPGA finishes configuration. The green LED indicates that the PCI Express interface link is active.

LED	Ref Des	Function
Blue	**D1**	**ON = Module power is ready for use: power is good and the FPGA is configured.**
Green	**LD2**	**ON = PCI Express link is up**
Red	**LD1**	**No specific use.**

Table 28. X6 LED Indicators

Custom logic designs can use it for any purpose. When using the stock firmware, the state of user logic LEDs can be controlled using the `Innovative::X6_400M::Led()` property.

LEDs NOT Lit with FrameWork Logic Installed

If the two LEDs do not light up with the FrameWork Logic installed, that means the PCI Express bus did not connect. This is a bus error. The card cannot communicate with the system . Check installation and contact technical support.

JTAG Scan Path

X6 modules have a JTAG scan path for the Xilinx devices on the module. This is used for logic development tools such as Xilinx ChipScope and System Generator, and for initial programming of the PCI FPGA configuration FLASH ROM.

Nominally, there are two devices in the scan chain: the Virtex-6 device and the Coolrunner CPLD used to implement the configuration support. Optionally, the SRAM devices may be included in the scan chain if JTAG access is needed for debugging purposes.

JTAG Device Number	Device	Function
0	Virtex-6	Application logic
1	Coolrunner-II	Configuration control

Table 29. X6 JTAG Scan Path

The JTAG connector (JP3) pinout is shown in hardware chapter of this manual.

FrameWork Logic

Many of the standard X6 XMC features are implemented in the application logic. The logic allows the card to be used for data acquisition and playback, with supporting software for data analysis, logging and controls. The logic features include a data plane, triggering features, and application-specific functions. In this manual, the FrameWork Logic features for each card are described in in general to explain the standard hardware functionality.

The *X6 FrameWork Logic User Guide* provides developers with the tools and know-how for developing custom logic applications. See this manual and the supporting source code for more information. The X6 modules are supported by the FrameWork Logic Development tools that allow designs to be developed in HDL or MATLAB Simulink. Standard features are provided as components that may be included in custom applications, or further modified to meet specific design requirements.

Integrating with Host Cards and Systems

The X6 XMCs may be directly integrated PCI Express systems that support VITA 42.3 XMC modules. The host card must be both mechanically and electrically compatible or an adapter card must be used.

The XMC modules conform to IEEE 1386 specification for single width mezzanine cards . This specification is common to both PMC and XMC modules and specifies the size, mounting, mating card requirements for spacing and clearances.

There are several adapter cards that are used to integrate the XMC modules into other form-factor PCI Express systems, such as desktop systems.

There are also adapter cards to electrically adapter the PCI Express XMC modules in older PCI systems that use a bridge device between the two buses. PCI is not electrical

Host Type	Bus	Mechanical Form-factor	Adapter Required	Example card
XMC.3 module slot	PCI Express 1.0a	XMC, single width	None	Kontron CP6012 www.kontron.com Diversified Technology CPB4712 http://www.diversifiedtechnology.com/products/cpci/cpb4712.html
Desktop PC (PREFERRED FOR X6)	PCI Express 1.0a	PCI Express Plug-in card, x8 lane	PCIe-XMC.3 adapter	Innovative 80259
Desktop PC	PCI Express 1.0a	PCI Express Plug-in card, x8 lane	PCIe-XMC.3 adapter	Innovative 80173
Desktop PC	PCI 2.2	PCI Plug-in card	PCI-XMC.3 adapter	Innovative 80167
Compact PCI	PCI Express 1.0a	3U	CPCI-XMC.3 adapter	Innovative 80207
Cabled PCI Express	PCI Express 1.0a	Cabled PCI Express to remote IO	Cable PCIe Adapter and XMC.3 carrier	Innovative 90181
Embedded PC Host	PCI Express to local PC core	~10x7x3 inches	None	Innovative 90200 or 90201
VPX	VPX with PCI Express	3U VPX	VPX 3U adapter Air or conduction cooled	Innovative Air Cooled : 80260-7 Conduction Cooled: 80260-6RC

Table 30. XMC Adapters and Hosts

Standalone Operation

The X6 modules can be run “standalone” for embedded applications using a carrier card such as Innovative 90181. The host card provides power to the module and cooling. Custom carrier cards have been created for specific applications that provide Ethernet ports,

Figure 28. eInstrument Node Enclosure (P/N 90181) Supports Standalone Operation

The logic must be modified for standalone operation to remove the PCIe host interface and controls and substitute local controls and logic. Other changes vital to system operation are detailed in the Framework Logic manuals.

Logic Configuration

FPGA logic can be loaded from either a non-volatile FLASH EEPROM or through the FPGA JTAG port. The logic image is loaded from the FLASH through the SelectMap interface to the FPGA at power up for normal operation. The JTAG port is usually used during debugging and test. Use of the JTAG port is described in the *X6 FrameWork Logic User Guide.*

Logic Configuration from FLASH EEPROM

The FPGA loads its logic from a non-volatile FLASH EEPROM on the X6 module. The X6 module has a logic loader built on the module that loads the FPGA logic image from the FLASH memory after power-on. The image size is dependent on the FPGA device.

The logic loader reads data from the FLASH memory bank and writes it into the FPGA SelectMap interface. To achieve the required configuration times for PCI Express systems, the two FLASH memories are read together as a 32-bit word, which is split into two 16-bit words and written into the FPGA at 80 MHz. This architecture requires that the two FLASH devices are programmed as a 32-bit wide memory instead of two 16-bit devices. Therefore, both devices are used for configuration.

It is not recommended that the configuration FLASH be used for other purposes because of its unique purpose on the module. The serial FLASH memory is available for other data and programs and should be used for those purposes.

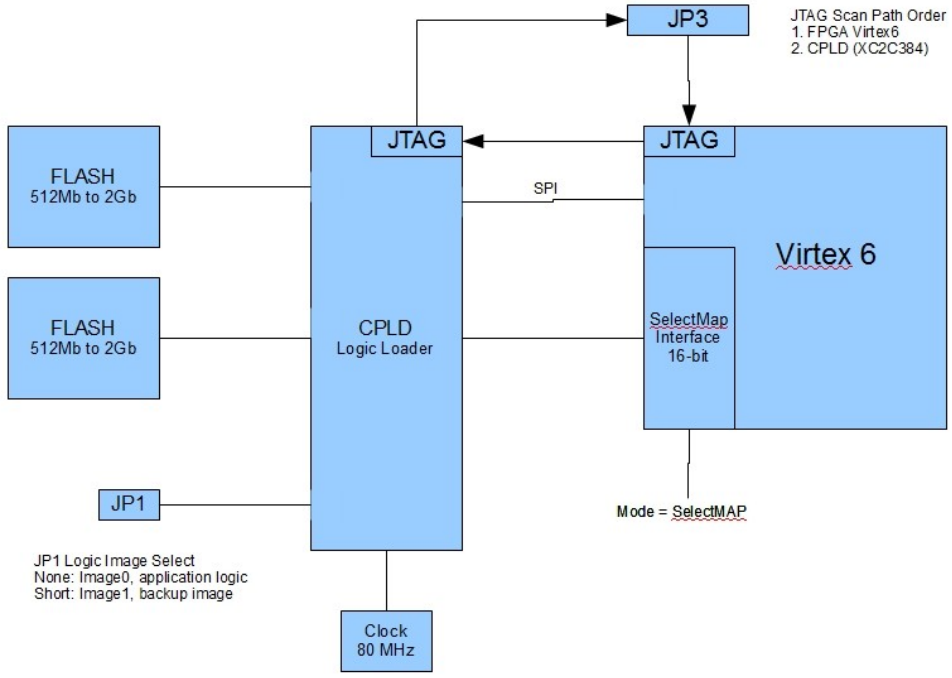


Figure 29. X6 Logic Configuration Subsystem

Device	Image Size (bits)	Time to Program (per image)	Time to Configure	Notes
XC6VLX240T	73,859,552	300 s	62 ms	
XC6VSX315T	104,465,888	430 s	86 ms	
XC6VLX475T	156,689,504	640 s	126 ms	

Table 31. X6 FPGA Logic Image Sizes and Configuration Times

Logic Image Selection

Two images are held in the FLASH memory at all times. The first image is the application logic, the second image is the backup image.

Image	JP1	Purpose	Notes
0	none	Application Logic	
1	Short	Backup Image	

Table 32. Logic Image Selection Jumper JP1

Updating the Logic Configuration FLASH EEPROM

Virtex-6 configuration data is stored in an onboard FLASH EEPROM which may be updated using software provided by Innovative. Logic images may come as updates from Innovative or be generated by a user developing custom functionality.

The applet provided by Innovative, VsProm.exe, programs the FLASH using a .bit or .exo image file generated by the Xilinx toolset.

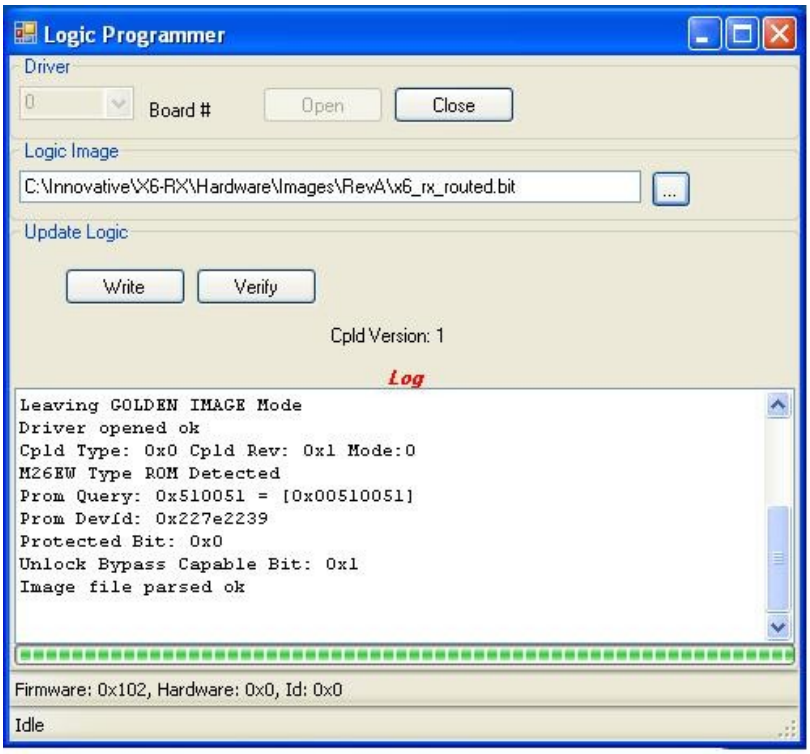


Figure 30. X6 Configuration FLASH Programmer

The EEPROM application is straightforward to use: a Target board is selected, then a .BIT file is selected for reprogramming. The Target number tells the software which XMC module to program. If you have multiple XMC modules in the system, each has a unique Target number assigned by the software. If you don't know which card is which target, you can use the Finder program to blink the LED on each Target.

Once you have selected the .BIT file, press the load button to begin programming. The progress bar shows that the programming is underway and when it is completed. Programming a few minutes due to the size of the configuration bitstreams used by the Virtex-6 device. DO NOT TURN OFF THE HOST COMPUTER OR RESTART IT UNTIL PROGRAMMING IS SUCCESSFULLY COMPLETED.

Once the EEPROM is reprogrammed, the X6 module must be power-cycled for reconfiguration to take place and the new bitstream loaded into the Virtex-6 device.

Rescuing the Card When the Logic Image is Bad

If an invalid image is programmed into the EEPROM, or the process is interrupted before completion for any reason, the Virtex-6 may no longer configure properly or may not communicate properly on the PCIe bus. If this occurs, then the card can be booted from the backup image or can be programmed using a JTAG cable.

The backup image, or "golden" image, may be used to boot the X6 card by installing a jumper on JP1. This forces the logic loader to load the X6 logic with the backup image. The card should boot with the backup image, then allowing the primary image to be reprogrammed using the EEPROM applet.

If the backup image is bad or not available, then the card can be programmed with a Xilinx JTAG using a known good image, followed by a warm-boot of the host system to allow the host OS to recognize the X6 module on the PCIE bus. Once this is accomplished, a known-good image may be reprogrammed into the EEPROM using the EEPROM.exe tool.

Chapter 4: *Writing Custom Applications*

Most scientific and engineering applications require the acquisition and storage of data for analysis after the fact. Even in cases where most data analysis is done in place, there is usually a requirement that some data be saved to monitor the system. In many cases a pure data that does no immediate processing is the most common application.

The X6 XMC card family contains high bandwidth analog capture and playback modules with an advanced architecture that provides ultimate flexibility and speed for the most advanced hardware-assisted signal processing and ultrasonic signal capture. The maximum data rate from these module are often in the hundreds of MSPS. This means that a simple logger that saves all of the data to the host disk is not feasible using standard operating system disk I/O calls, as the slower disk writes eventually cause overflow and data loss in the streaming system.

Some modules support decimation so that long duration samples can be taken without data. Also quick snapshots of analog data can be taken without loss as long as the amount of data is less than the net capacity of system memory and what the baseboard holds. The example program provided for nearly all cards is this limited capacity data logger, called the Snap example

The Snap Example

The Snap example in each software distribution demonstrates this logging functionality. It consists of a host program in Windows, which works with the logic provided on the board's flash to stream data to the host. It uses the Innovative Malibu software libraries to accomplish the tasks.

Tools Required

In general, writing applications for the XMC requires the development of host program. This requires a development environment, a debugger, and a set of support libraries from Innovative.

Table 33. Development Tools for the Windows Snap Example

Processor	Development Environment	Innovative Toolset	Project Directory
Host PC	Codegear Developers Studio C++	Malibu	Examples\Snap\Bcb11
	Microsoft Visual Studio		Examples\Snap\VC9
	QtCreator		Examples\Snap\Qt
	Common Host Code		Examples\Snap\Common

On the host side, the Malibu library is provided in source form, plus pre-compiled Microsoft, Borland or GCC libraries. The application code that implements the entirety of the board-specific functionality of example is factored into the `ApplicationIo.cpp/h` unit. All user interface aspects of the program are completely independent from the code in

`ApplicationIo`, which contains code portable to any compilation environment (i.e., it is common code). While each compiler implements the GUI differently, each version of the example project uses the same code file to interact with the hardware and acquire data.

Program Design

The Snap example is designed to allow repeated data reception operations on command from the host. As mentioned earlier, received data can be saved as Host disk files. When using modest sample rates, data can be logged to standard disk files. However, full bandwidth storage of multiple A/D channels can require up more capacity, so a dedicated RAID0 drive array for data storage may be required, or data may have to be cached online and stored after stopping data flow. The example application software is written to perform minimal processing of received data and is a suitable template for high-bandwidth applications.

The example uses various configuration commands to prepare the module for data flow. Parametric information is obtained from a Host GUI application, but the code is written to be GUI-agnostic. All board-specific I/O is performed within the `ApplicationIo.cpp/.h` unit. Data is transferred from the module to the Host as packets of `Buffer` class objects.

The Host Application

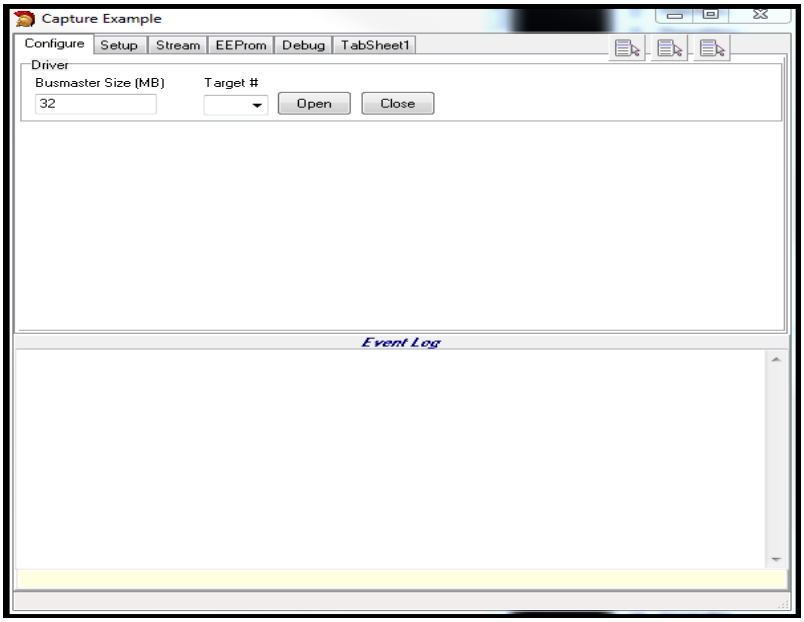
The picture to the right shows the main window of an X6 example (for the X6-RX). This form is from the designer of the CodeGear Cbuilder 2007 version of the example, but the Microsoft and Qt versions are similar. It shows the layout of the controls of the User Interface.

User Interface

This application has five tabs. Each tab has its own significance and usage, though few are interrelated. All these tabs share a common area, which displays messages and feedback throughout the operation of the program.

Configure Tab

As soon as the application is launched, the Configure tab is displayed. In this tab, a combo box is available to allow the selection of the device from those present in the system. All XMC family devices of whatever type share a sequence of target number identifiers. The first board found is Target 0, the second Target 1, and so on.

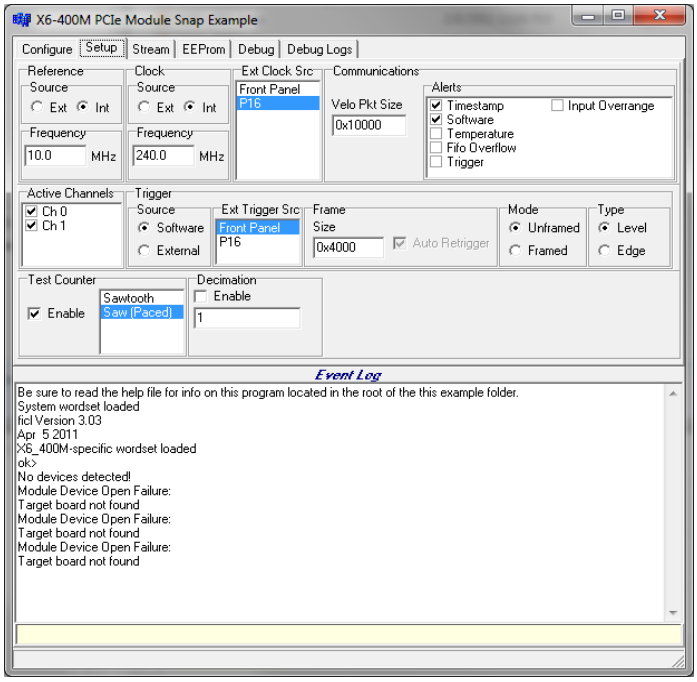


Click the `Open` button to open the driver. To change targets, click the `Close` button to close the driver, select the number of the desired target using the `Target #` combo box, then click `Open` to open communications with the specified target module. The order of the targets is determined by the location in the PCI bus, so it will remain unchanged from run to run unless the board is moved to a different slot or another target is installed.

Setup Tab

This tab has a set of controls that hold the parameters for transmission. These settings are delivered to the target and configure the target accordingly. This tab has several sections.

The controls in the Clock group offer configurations and routing of the clock. The clock for the FPGA can come from an external clock (EXT) or from an onboard phase-locked loop (INT). The selection can be made at upper right corner of this section. The output sample rate of the selected clock source is specified in the Output field in MHz.



The controls in the Reference group specify the source and frequency of the reference clock. The behavior of the reference clock and the meaning of these controls are a function specified Clock Source specified above. If the clock source is internal, the Reference controls specify the source and rate of the reference clock routed to the onboard PLL. If the clock source is external, the Reference controls specify the source and rate of the sample clock, bypassing the PLL. In this mode, the ratio of the Reference Frequency/Clock Frequency is used to control an onboard clock divider, supporting sampling at integer submultiple rates of the reference clock. The table below summarizes the behavior.

The physical line driving the external clock/reference signal has a multiplexer on it that can select one of two sources for the signal. One is the front panel input, the second choice is to use the P16 connector clock as the source. This signal is driven by Innovative's SBC-Comex product line.

Reference Source	Clock Source	PLL Reference	Sample Clock	Sample Rate
EXT	EXT	N/A	EXT CLK	External rate / (REF_FREQ/CLK_FREQ)
INT	EXT	N/A	Onboard Oscillator	100 MHz / (REF_FREQ/CLK_FREQ)
EXT	INT	EXT CLK	PLL	CLK_FREQ
INT	INT	Onboard Oscillator	PLL	CLK_FREQ

The Communications section controls the Alert features and the input data packets size. Checking the box next to an alert will allow the logic to generate an alert if that condition occurs during data streaming. This alert can then be left in the data stream, or extracted to notify the application. The Velo Packet Size edit control specifies the size of the Velocia packets that the module will use to

In the Channels section, we can specify number of channels to activate. Selecting a channel will flow data from that data source during streaming.

The Trigger group contains controls that affect the way that I/O starts and stops. Triggers act as a gate for I/O - no data flows until a trigger has been received. Triggers may be initiated via software or hardware, depending on the Trigger | Source control. If software, the application program must issue a command to initiate data flow. If hardware, a signal applied to the external trigger connector controls data flow. The external trigger line may also be routed from one of two sources, either the front panel connector or the P16 clock signal line.

Triggers are modal depending on the Trigger | Mode control. In Unframed mode, triggers are level sensitive and data flow proceeds while the trigger is in the high (active) state and stops while the trigger is in the low (inactive) state. This mode is appropriate for conventional data acquisition applications. In Framed mode, triggers can be selected to be edge sensitive or level sensitive on the X6. Upon detection of each trigger, Trigger | Frame | Count samples are acquired from all active channels, then acquisition terminates until the next trigger edge is detected. This mode is well-suited for applications such as spectral analysis using fixed input buffers submitted to FFTs.

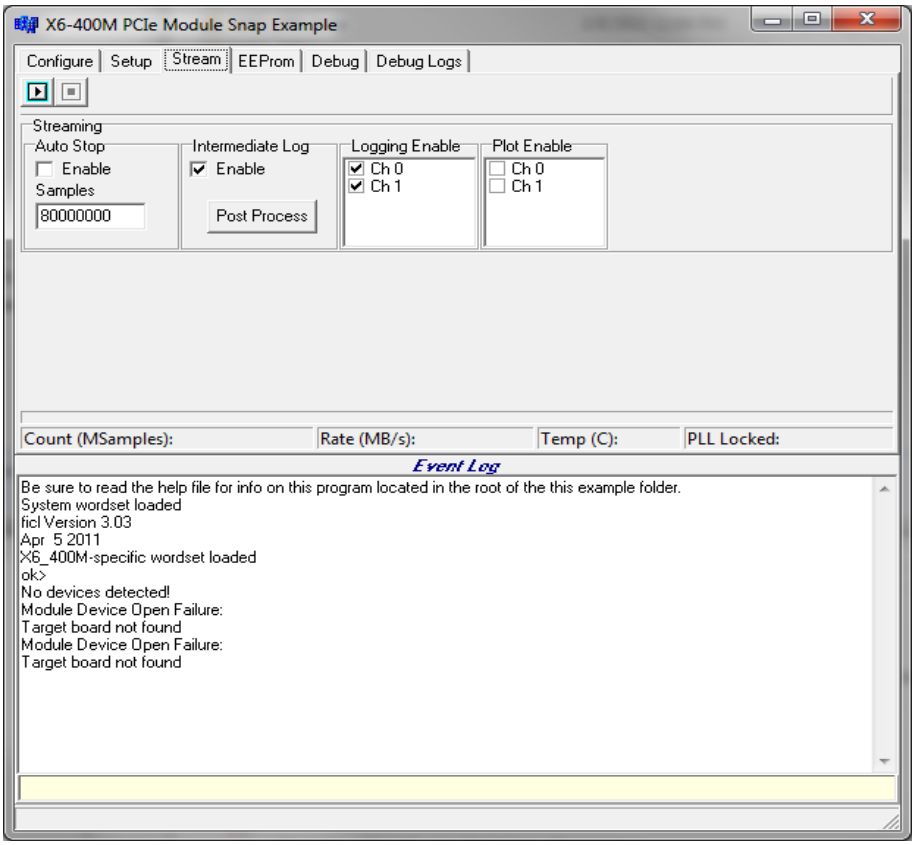
The module supports one or more test modes for module debugging and system test. The Test Counter Enable is used to turn on test mode and substitutes a ramp signal with channel number in the upper byte of the data. The Decimation section sets up the decimation logic to discard data, reducing the incoming data rate.

Stream Tab

The two buttons in the button bar start and stop data streaming. Press the running man button to start streaming data. Press the stop button to stop streaming, unless the stream has stopped itself. When streaming, the status bar data is collected and displayed. This includes a count of the data samples received, the data rate, the measured temperature of the board logic, and the PLL locked indicator bit value.

Data is logged to an intermediate file that holds the Velocia buffers with the Vita Buffer data. This can be read by Binview in native mode for analysis. The Post Processing button will extract the data from the Intermediate file and convert it into X5 compatible data files for use with older versions if desired.

The Auto Stop option determines if data streaming stops after collecting the given number of points (a Snapshot of data) or continues to stream forever until manually stopped.



Host Side Program Organization

The Malibu library is designed to be built in numerous different host environments: Borland/CodeGear/Embarcadero CBuilder, Microsoft Visual Studio under Windows or QtCreator under Windows/Linux. Because the library is built using and identical source code base in all environments, the code that interacts with Malibu is identical regardless of the toolchain used. Within examples, the portion of the code which interacts with Malibu is factored into a class, *ApplicationIo* in the files ApplicationIo.cpp and .h. This class acts identically on all frameworks and OS platforms.

It is very important to realize that the most expedient method to create a custom application, regardless of the framework type used (.NET, VCL, Qt, wxWidgets or console mode), is to incorporate the ApplicationIo class in its entirety directly into your custom app. ApplicationIo contains only portable code, without any framework dependencies whatsoever. It contains all of the essential hardware initialization code, event handlers, etc to support efficient communications with the board. You'll likely need all of the features of ApplicationIo in your application - so why reinvent the wheel?

Many users are initially unclear on how ApplicationIo can remain agnostic to the framework. This is accomplished by factoring all functions used to relay information between framework-specific code and the ApplicationIo into a small separate class called UserInterface within the supplied examples.

The concrete UserInterface object derives from the pure abstract IUserInterface base class. IUserInterface simply specifies the functions that must be supplied by a user application to provide a bridge to the stock features of the ApplicationIo object. Your application must create a UserInterface class and pass it to the constructor of the ApplicationIo object, but your implementation of the IUserInterface methods may degenerate to trivial no-ops if desired.

The Main form of the application creates an *ApplicationIo* to perform the work of the example. The UI can call the methods of the *ApplicationIo* to perform the work when, for example, a button is pressed or a control changed.

Sometimes, however, the *ApplicationIo* object needs to 'call back into' the UI. But since the code here is common, it can't use a pointer to the main window or form, as this would make *ApplicationIo* have to know details of Borland or the VC environment in use.

The standard solution to decouple the *ApplicationIo* from the form is to use an Interface class to hide the implementation. An interface class is an abstract class that defines a set of methods that can be called by a client class (here, *ApplicationIo*). The other class produces an implementation of the Interface by either multiple inheriting from the interface, or by creating a separate helper class object that derives from the interface. In either case the implementing class forwards the call to the UI form class to perform the action. *ApplicationIo* only has to know how to deal with a pointer to a class that implements the interface, and all UI dependencies are hidden.

The predefined *IUserInterface* interface class is defined in ApplicationIo.h. The constructor of *ApplicationIo* requires a pointer to the interface, which is saved and used to perform the actual updates to the UI inside of *ApplicationIo's* methods.

ApplicationIo

Initialization

The main form creates an ApplicationIo object in its constructor. The object creates a number of Malibu objects at once as can be seen from this detail from the header ApplicationIo.h.

```
ModuleIo                          Module;
IUserInterface *                   UI;
Innovative::VitaPacketStream       Stream;
Innovative::VitaPacketParser       Vpp;
Innovative::TriggerManager         Trig;
Innovative::SoftwareTimer          Timer;
MultiLoggerManager                 MLM;
Innovative::StopWatch              RunTimeSW;
Innovative::DataLogger             IntermediateLogr;
Innovative::DataPlayer             IntermediatePlayer;

//  App State variables
bool                               Opened;
bool                               StreamConnected;
bool                               Stopped;
//  App Status variables
double                             FBlockRate;
ii64                               FWordCount;
int                                SamplesPerWord;
ii64                               WordsToLog;

Innovative::AveragedRate           Time;
Innovative::AveragedRate           BytesPerBlock;
     ...
```

In Malibu, objects are defined to represent units of hardware as well as software units. The ModuleIo object represents the board. The VitaPacketStream object encapsulates supported, board-specific operations related to I/O Streaming. The VitaPacketParser object encapsulates capabilities to extract VITA49 packets from the native Velocia packets supplied by the

board during streaming. The MultiLoggerManager object provides file storage for each peripheral type parsed from the stream by the parser, facilitating analysis of data received.

The ApplicationIo object derives from type FiclIo. FICL is an abbreviation for Forth-Inspired Control Language. A simple Forth interpreter is available within the module object which can be used as a simple scripting language, for the purposes of performing hardware initialization during FPGA firmware development.

When the Open button is pressed, the ApplicationIo object begins the process of opening the board driver and setting up the board for a run. The first thing done is to link Malibu software events to callback functions in the applications by setting the handler functions.

Malibu uses *events* to allow functions to be 'plugged into' the library to be called at certain times or in response to certain events detected. Events allow a tight integration between an application and the library:

```
//
//  Configure Trigger Manager Event Handlers
Trig.OnDisableTrigger.SetEvent(this, &ApplicationIo::HandleDisableTrigger);
Trig.OnExternalTrigger.SetEvent(this, &ApplicationIo::HandleExternalTrigger);
Trig.OnSoftwareTrigger.SetEvent(this, &ApplicationIo::HandleSoftwareTrigger);
```

This code attaches event handlers associated with the various trigger conditions. For example, the OnDisableTrigger event fires when triggering is disabled. When Malibu detects that condition, it calls the ApplicationIo::HandleDisableTrigger method, providing a means of user-specific processing under that condition.

In a similar manner, the events hooked below are called at strategic times during data streaming:

```
//
//  Configure Module Event Handlers
Module().OnBeforeStreamStart.SetEvent(this, &ApplicationIo::HandleBeforeStreamStart);
Module().OnBeforeStreamStart.Synchronize();
Module().OnAfterStreamStart.SetEvent(this, &ApplicationIo::HandleAfterStreamStart);
Module().OnAfterStreamStart.Synchronize();
Module().OnAfterStreamStop.SetEvent(this, &ApplicationIo::HandleAfterStreamStop);
Module().OnAfterStreamStop.Synchronize();
```

HandleBeforeStreamStart, HandleAfterStreamStart and HandleAfterStreamStop handle events issued on before stream start, after stream start and after stream stop respectively. These handlers could be designed to perform multiple tasks as event occurs including displaying messages for user. These events are tagged as Synchronized, so Malibu will marshal the execution of the handlers for these events into the main thread context, allowing the handlers to perform user-interface operations.

The hooking of alerts are factored into a method within the Module object, since different modules may provide different alert capabilities.

```
//  Alerts
Module.HookAlerts();
```

This code attaches alert processing event handlers to their corresponding events. Alerts are packets that the module generates and sends to the Host as packets containing out-of-band information concerning the state of the module. For instance, if the analog inputs were subjected to an input over-range, an alert packet would be sent to the Host, interspersed into the data stream, indicating the condition. This information can be acted upon immediately, or simply logged along with analog data for subsequent post-analysis.

```
//
//  Configure Stream Event Handlers
Stream.OnVeloDataAvailable.SetEvent(this, &ApplicationIo::HandleDataAvailable);
Stream.DirectDataMode(false);
```

The Stream object manages communication between the application and a piece of hardware. Separating the I/O into a separate class clarifies the distinction between an I/O protocol and the implementing hardware.

In Malibu, high rate data flow is controlled by one of a number of streaming classes. In this example we use the events of the VitaPacketStream class to alert us when a packet arrives from the target. When a data packet is delivered by the data streaming system, OnDataAvailable event will be issued to process the incoming data. This event is set to be handled by HandleDataAvailable. After processing, the data will be discarded unless saved in the handler. Similarly, "OnDataRequired" event is handled by HandleDataRequired. In such a handler, packets would be filled with data for output to the baseboard. The Snap application does not generate output, so the HandleDataRequired event is left unhandled.

```
Timer.OnElapsed.SetEvent(this, &ApplicationIo::HandleTimer);
Timer.OnElapsed.Thunk();
```

In this example, a Malibu SoftwareTimer object has been added to the ApplicationIo class to provide periodic status updates to the user interface. The handler above serves this purpose.

An event is not necessarily called in the same thread as the UI. If it is not, and if you want to call a UI function in the handler you have to have the event synchronized with the UI thread. A call to Synchronize() directs the event to call the event handler in the main UI thread context. This results in a slight performance penalty, but allows us to call UI methods in the event handler freely. The Timer uses a similar synchronization method, Thunk(). Here the event is called in the main thread context, but the issuing thread does not wait for the event to be handled before proceeding. This method is useful for notification events.

Creating a hardware object does not attach it to the hardware. The object has to be explicitly opened. The Open() method of the baseboard activates the board for use. It opens the device driver for the baseboard and allocates internal resources for use. The next step is to call Reset() method which performs a board reset to put the board into a known good state. Note that reset will stop all data streaming through the busmaster interface and it should be called when data taking has been halted.

The size of the busmaster region is changeable by using the BusMasterSize() property before opening the board. Larger values provide more buffering during streaming, at the cost of slower allocation at startup time. Under Windows, up to 32 MB can generally be allocated in each stream direction. Under Linux, only up to 4 MB can generally be allocated in each stream direction.

```
// Insure BM size is a multiple of four MB
const int Meg = 1024 * 1024;
const int BmSize = std::max(Settings.BusmasterSize/4, 1) * 4;
Module().IncomingBusMasterSize(BmSize * Meg);
Module().OutgoingBusMasterSize(1 * Meg);
Module().Target(Settings.Target);
//
//  Open Device
try
    {
    Module().Open();

    std::stringstream msg;
    msg << "Bus master size: " << BmSize << " MB";
    Log(msg.str());
    }
```

```
catch(Innovative::MalibuException & e)
    {
    UI->Log("Module Device Open Failure:");
    UI->Log(e.what());
    return;
    }

catch(...)
    {
    UI->Log("Module Device Open Failure: Unknown Exception");
    Opened = false;
    return;
    }

Module().Reset();
UI->Log("Module Device opened successfully...");
Opened = true;
```

This code shows how to open the device for streaming. Each baseboard has a unique code given in a PC. For instance, if there are three boards in a system, they will be targets 0, 1 and 2. The order of the targets is determined by the location in the PCIe bus, so it will remain unchanged from run to run. Moving the board to a different PCIe slot may change the target identification. The Led() property can be use to determine the association between a target number and the physical board in a configuration.

```
    //
    //  Connect Stream
    Stream.ConnectTo(&(Module.Ref()));
    StreamConnected = true;
    UI->Status("Stream Connected...");
```

Once the object is attached to actual physical device, the streaming controller associates with a baseboard by the ConnectTo() method. The variable Module is a class, not a baseboard, to allow quicker porting. The Module.Ref() is the admittedly ugly way of getting a pointer to the internal Module pointer for connection – Ref returns a reference, and the address-of operator (&) turns this into a pointer.

Once connected, the object is able to call into the baseboard for board-specific operations during data streaming. If an objects supports a stream type, this call will be implemented. Unsupported stream types will not compile.

Lastly we capture and display some information to the screen. This includes the logic version, PCI bus version information, etc.

```
    DisplayLogicVersion();
}
```

Similarly, the Close() method closes the hardware. Inside this method, first we logically detach the streaming subsystem from its associated baseboard using Disconnect() method. Malibu method Close() is then used to detach the module from the hardware and release its resources.

```
//-----------------------------------------------------------------------------
//  ApplicationIo::Close() -- Close Hardware & set up callbacks
//-----------------------------------------------------------------------------

void  ApplicationIo::Close()
{
    if (!Opened)
        return;

    Stream.Disconnect();
```

```
    StreamConnected = false;

    Module().Close();
    Opened = false;

    UI->Log("Stream Disconnected...");
}
```

Starting Data flow

After downloading interface logic user can setup clocking and triggering options. The stream button then can be used to start streaming and thus data flow.

```
//-------------------------------------------------------------------------
//  ApplicationIo::StartStreaming() --  Initiate data flow
//-------------------------------------------------------------------------

void  ApplicationIo::StartStreaming()
{
    if (!StreamConnected)
        {
        Log("Stream not connected! -- Open the boards");
        return;
        }

    //
    //  Set up Parameters for Data Streaming
    //  ...First have UI get settings into our settings store
       UI->GetSettings();
```

Before we start streaming, all necessary parameters must be checked and loaded into option object. UI-> GetSettings() loads the settings information from the UI controls into the Settings object within the ApplicationIo class.

```
if (Settings.SampleRate*1.e6 > Module().Input().Info().MaxRate())
    {
    Log("Sample rate too high.");
    StopStreaming();
    UI->AfterStreamAutoStop();
    return;
    }
```

We insure that the sample rate specified by the GUI is within the capabilities of the module.

```
if (Settings.Framed)
    {
    // Granularity is firmware limitation
    int framesize = Module().Input().Info().TriggerFrameGranularity();
    if (Settings.FrameSize % framesize)
        {
        std::stringstream msg;
        msg << "Error: Frame count must be a multiple of " << framesize;
        Log(msg.str());
        UI->AfterStreamAutoStop();
        return;
        }
    }
```

The module supports both framed and continuous triggering. In framed mode, each trigger event, whether external or software initiated, results in the acquisition of a fixed number of samples. In continuous mode, data flow continues whenever the trigger is active, and pauses while the trigger is inactive. The code above issues a warning if the trigger mode is framed and ill-formed.

```
FWordCount = 0;
unsigned int SamplesPerWord = Module().Input().Info().SamplesPerWord();
WordsToLog = Settings.SamplesToLog / SamplesPerWord;

FBlockRate = 0;
```

The class variables above are used to maintain counts of blocks received, reception rate and whether the module is currently triggered. These values are initialized prior to each streaming run.

```
// Disable triggers initially  (done by library)
//  Set external trigger?
Trig.AtConfigure();
//
//  Channel Enables
Module().Input().ChannelDisableAll();
for (unsigned int i = 0; i < Module().Input().Channels(); ++i)
    {
    bool active = Settings.ActiveChannels[i] ? true : false;
    if (active==true)
        Module().Input().ChannelEnabled(i, true);
    }
```

The number of channels supported varies from board to board. The code uses standard functions like Module().Input().Channels() to obtain the max channel count from a board. The ModuleIo also has a standard function AnalogInChannels() that can be used before the board is opened and other methods are valid. The previous call to GetSettings populated the Settings object with the number of channels to be enabled on this run. That information is used to enable the required channels via the Channels object within the Module.Input() object.

The clock source is also programmed using the associated methods within the Module object:

```
// Route reference.
IX6ClockIo::IIReferenceSource ref[] = { IX6ClockIo::rsExternal, IX6ClockIo::rsInternal };
Module().Clock().Reference(ref[Settings.ReferenceClockSource]);
Module().Clock().ReferenceFrequency(Settings.ReferenceRate * 1e6);

// Route clock
IX6ClockIo::IIClockSource src[] = { IX6ClockIo::csExternal, IX6ClockIo::csInternal };
Module().Clock().Source(src[Settings.SampleClockSource]);
Module().Clock().Frequency(Settings.SampleRate * 1e6);

// Readback Frequency
double freq_actual = Module().Clock().FrequencyActual();
{
std::stringstream msg;
msg << "Actual PLL Frequency: " << freq_actual ;
Log(msg.str());
}
```

The size of the data packets sent from the module to the Host during streaming is programmable. This is helpful during framed acquisition, since the packet size can be tailored to match a multiple of the frame size, providing application notification on each acquired frame. In other applications, such as when an FFT is embedded within the FPGA, the packet size can be programmed to match the processing block size from the algorithm within the FPGA.

```
//
//  Always preconfigure
Stream.Preconfigure();
```

In order to support modes where the clock is not disturbed from run to run, some configuration (notably clock configuration) is performed in a preconfiguration step. Here we call this step automatically to program the clock for the run.

```
//
//  Velocia Packet Size
Module.SetInputPacketDataSize(Settings.PacketSize);
```

The I/O data can be replaced with FPGA-generated test data by enabling one of the test modes. This is useful when developing custom logic. The code snippet below applies the test mode stored in the Settings.TestGenMode variable to the hardware:

```
//
//  Input Test Generator Setup
Module.SetTestConfiguration( Settings.TestCounterEnable, Settings.TestGenMode );
```

I/O samples can be decimated using a feature within the stock firmware. The code snippet below applies the specified decimation factor to the hardware:

```
// Set Decimation Factor
int factor = Settings.DecimationEnable ? Settings.DecimationFactor : 0;
Module().Input().Decimation(factor);
```

The streamed data is logged without change to an intermediate file that can be analyzed with BinView. If this logging is enabled, this starts the logger.

```
    if (Settings.IntermediateLogEnable)
        {
        IntermediateLogr.Start();
        }
```

The code below applies the user-specified trigger and and alert configuration settings to the hardware.

```
// Frame Triggering and other Trigger Config
Module().Input().Trigger().FramedMode(Settings.Framed);
Module().Input().Trigger().Edge(Settings.EdgeTrigger);
Module().Input().Trigger().FrameSize(Settings.FrameSize);

Module.ConfigureAlerts(Settings.AlertEnable);

Trig.AtStreamStart();
```

Samples will not be acquired until the channels are triggered. Triggering may be initiated by a software command or via an external input signal to the Trigger SMA connector. The module supports framed triggering, where a single trigger enables many data samples to be taken before rechecking the trigger. This code enables framed mode, or disables it depending on the settings.

The Stream Start command applies all of the above configuration settings to the module, then enables PCI data flow. The software timer is then started as well.

```
    //
    //  Start Streaming
```

```
    Stream.Start();
    UI->Log("Stream Mode started");
    UI->Status("Stream Mode started");

    FTicks = 0;
    Timer.Enabled(true);
}
```

Handle Data Available

Once streaming is enabled and the module is triggered, data flow will commence. Samples will be accumulated into the onboard FIFO, then they are bus-mastered to the Host PC into page-locked, driver-allocated memory following a two -word header (data packets). Upon receipt of a data packet, Malibu signals the Stream.OnDataAvailable event. By hooking this event, your application can perform processing on each acquired packet. Note, however, that this event is signaled from within a background thread. So, you must not perform non-reentrant OS system calls (such as GUI updates) from within your handler unless you marshal said processing into the foreground thread context. This marshaling behavior can be automatically enabled by calling the event Thunked() or Synchronized() method, as was done in the Open() call earlier.

```
//---------------------------------------------------------------------------
//  ApplicationIo::HandleDataAvailable() --  Handle received packet
//---------------------------------------------------------------------------

void  ApplicationIo::HandleDataAvailable(VitaPacketStreamDataEvent & Event)
{
    if (Stopped)
       return;

    VeloBuffer Packet;

    //
    //  Extract the packet from the Incoming Queue...
    Event.Sender->Recv(Packet);

    FWordCount += Packet.SizeInInts();

    if (Settings.IntermediateLogEnable)
        if (FWordCount < WordsToLog)
            {
            IntermediateLogr.LogWithHeader(Packet);
            }

    IntegerDG Packet_DG(Packet);

    TallyBlock(Packet_DG.size()*sizeof(int));

    // Per block triggering actions
    Trig.AtBlockProcess(Packet_DG.size()*sizeof(int));
}
```

When the event is signaled, the data buffer must be copied from the system bus-master pool into an application buffer. The preceding code copies the packet into the local Buffer called Packet. Since data sent from the hardware can be of arbitrary type (integers, floats, or even a mix, depending on the board and the source), Buffer objects have no assumed data type and have no functions to access the data in them. In the case of X6 series XMC modules, the data contained within the body of the buffer is a list of VITA49 subpackets. These must be parsed and processed individually. This is done as a post-processing step, since in normal cases the Binview application can display the contents of the data file without splitting it up.

As each Velocia packet is received and processed within the HandleDataAvailable method, the TallyBlock method is called. This routine calculates and reports the data flow rate through the user interface.

```
//---------------------------------------------------------------------------
//  ApplicationIo::TallyBlock() --  Finish processing of received packet
//---------------------------------------------------------------------------

void  ApplicationIo::TallyBlock(size_t bytes)
{
    double Period = Time.Differential();
    double AvgBytes = BytesPerBlock.Process(static_cast<double>(bytes));
    if (Period)
        FBlockRate = AvgBytes / (Period*1.0e6);

...
}
```

In this example, each received packet is logged to a disk file. The packet header and the body are written into the file, which implies that a post-analysis tool (such as BinView) must be used to parse packetized data from the file. Alternately, custom applications may use the Innovative::VitaPacketFileDataSet object to conveniently extract channelized data from a Vita formatted packet data source. Packets are processed until a specified amount of data is logged or the GUI Stop button is pressed.

```
    //
    //  Stop streaming when both Channels have passed their limit
    if (Settings.AutoStop && IsDataLoggingCompleted() && !Stopped)
        {
        // Stop counter and display it
        double elapsed = RunTimeSW.Stop();

        StopStreaming();
        Log("Stream Mode Stopped automatically");
        Log(std::string("Elasped (S): ") + FloatToString(elapsed));
        }
}
```

The Wave Example

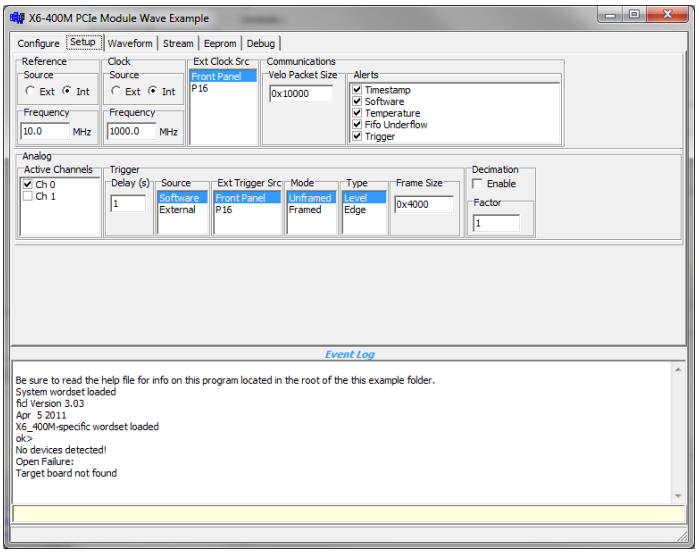
The Wave example in the software distribution demonstrates output streaming. It will only be included if the board supports streaming out to a DAC or similar device or devices.

In many ways the Wave example is similar to the Snap example. Differences are highlighted in this section.

User Interface

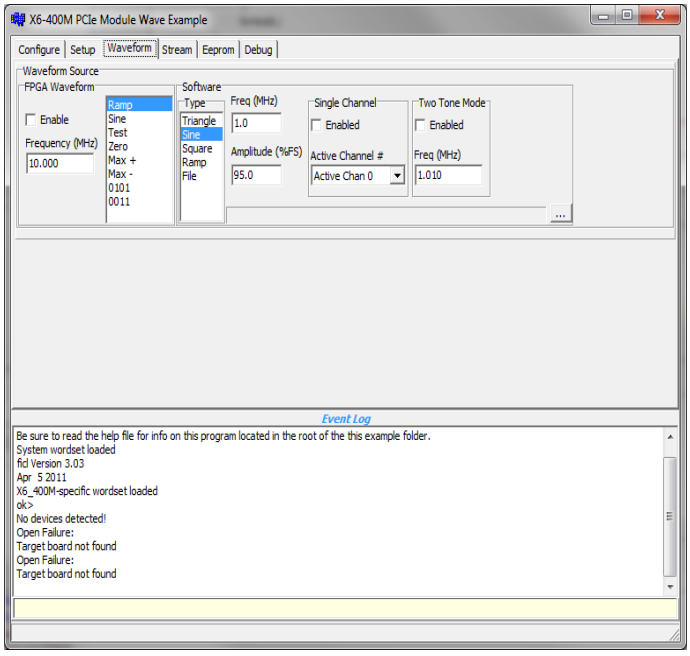
The Setup Tab

This tab, like that for the Snap application, allows the user to set up the configuration of the board before starting streaming. An additional option on this tab is the Trigger Delay setting. This postpones the software start trigger for a given time to allow the application ample time to stream data into the card before starting output.



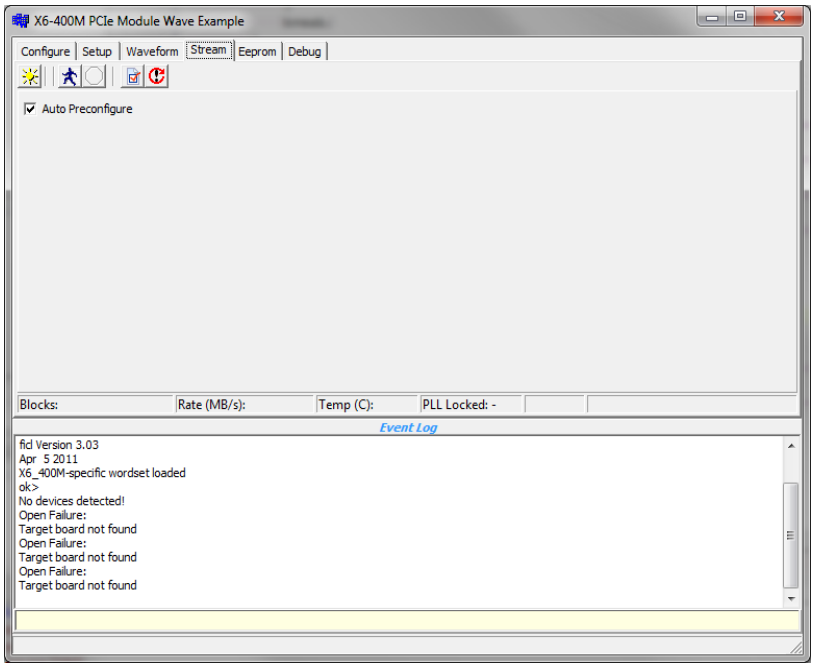
The Waveform Tab

This tab collects the configuration of the output waveform into one area. If the FPGA Waveform is enabled, then it plays the time of waveform selected. If a Sine wave is chosen, then the frequency value is used as the rate of the sine wave. In this mode, no data is streamed by the software.



The software waveform generator uses the configuration selected to generate a single buffer's worth of data (indicated by the size of the Velocia packet on the setup tab). All active channels are filled with the same data, unless single channel is enabled. In that case the indicated channel of those active is filled with data, and the remainder are zero filled. Two tone mode fills the buffer with the sum of two waves of the indicated frequencies.

Stream Tab
The Wave example supports the Preconfigure button. This allows the DAC timebase and calibration to occur once before starting streaming, leaving the clock undisturbed during the run. If the Auto Preconfig checkbox is checked, then it acts as if you pressed the Preconfigure button every run.



ApplicationIo

Stream Preconfigure
In order to support DAC systems that need calibration to tune the synchronization of their outputs, the channel selection and timebase definition is moved out of the 'normal' location to a special preconfiguration stage. This stage must be manually triggered by the application by calling Stream.Preconfigure(). The Wave example allows this by the additon of a preconfiguration button. In addition, the Auto Preconfigure mode will automatically call Stream.Preconfigure() whenever the stream is started.

All items that are initialized in the preconfiguration need to have their settings applied to the board object here. In general, the items that need to be set up are the clock/timebase configuration and the DAC configuration, particularly the enables.

```
//-----------------------------------------------------------------------------
// ApplicationIo::StreamPreconfigure()
//-----------------------------------------------------------------------------

void ApplicationIo::StreamPreconfigure()
{
    Log("Preconfiguring Stream....");
    //
    //  Make sure if Wave Generator is in single channel mode, we have a valid
    //    active channel filled in
    if (Builder.Settings.SingleChannelMode)
        {
```

```
            int active_channels = 0;
            for (unsigned int i = 0; i < Settings.ActiveChannels.size(); ++i)
                if ((Settings.ActiveChannels[i] ? true : false))
                    active_channels++;
            if (active_channels==3)
                active_channels = 4;   // can't do a true 3 channel run, promote to 4.
            if (Builder.Settings.SingleChannelChannel >= active_channels)
                {
                Log("Error: Invalid Active Channel selected in Single Channel Mode");
                return;
                }
            }
        //
        //  Set Channel Enables
        Module().Output().ChannelDisableAll();
        for (unsigned int i = 0; i < Module().Output().Channels(); ++i)
            {
            bool active = Settings.ActiveChannels[i] ? true : false;
            if (active==true)
                Module().Output().ChannelEnabled(i, true);
            }
        //
        // Clock Configuration
        //   Route ext clock source
        IX6ClockIo::IIClockSelect cks[] = { IX6ClockIo::cslFrontPanel, IX6ClockIo::cslP16 };
        Module().Clock().ExternalClkSelect(cks[Settings.ExtClockSrcSelection]);
        //   Route reference.
        IX6ClockIo::IIReferenceSource ref[] = { IX6ClockIo::rsExternal, IX6ClockIo::rsInternal };
        Module().Clock().Reference(ref[Settings.ReferenceClockSource]);
        Module().Clock().ReferenceFrequency(Settings.ReferenceRate * 1e6);
        //   Route clock
        IX6ClockIo::IIClockSource src[] = { IX6ClockIo::csExternal, IX6ClockIo::csInternal };
        Module().Clock().Source(src[Settings.SampleClockSource]);
        Module().Clock().Frequency(Settings.SampleRate * 1e6);
        // Readback Frequency
        double freq_actual = Module().Clock().FrequencyActual();
        {
        std::stringstream msg;
        msg << "Actual PLL Frequency: " << freq_actual ;
        Log(msg.str());
        }

        Stream.Preconfigure();
    }
```

Start Streaming

Setup of the stream is much like the Snap example. We have similar error checking code and rate guarding.

```
//---------------------------------------------------------------------------
// ApplicationIo::StartStreaming()
//---------------------------------------------------------------------------

void ApplicationIo::StartStreaming()
{
    //
    //  Set up Parameters for Data Streaming
    //  ...First have UI get settings into our settings store
    UI->GetSettings();

    //  if auto-preconfiging, call preconfig here.
    if (Settings.AutoPreconfig)
```

```
    StreamPreconfigure();

if (!FStreamConnected)
    {
    Log("Stream not connected! -- Open the boards");
    return false;
    }
if (Settings.SampleRate*1.e6 > Module().Output().Info().MaxRate())
    {

    Log("Sample rate too high.");
    return false;
    }
```

The first difference is that we configure the Output() sub-object instead of Input(). The X6 Family divides the interface functions for Input and Output devices into separate configuration sub-objects. This allows Input and Output to be independently configured.

```
if (Settings.Framed)
    {
    // Granularity is firmware limitation
    int framesize = Module().Output().Info().TriggerFrameGranularity();
    if (Settings.FrameSize % framesize)
        {
        std::stringstream msg;
        msg << "Error: Frame count must be a multiple of " << framesize;
        Log(msg.str());
        UI->AfterStreamStop();
        return false;
        }
    }
```

This code checks that the frame size is valid for the particular board being used. The granularity of the hardware frame size control could be different on different cards. But as long as there is a standard means of getting the correct value, the example can work for any card with the same code.

```
// Configure Trigger Mananger
Trig.DelayedTriggerPeriod(Settings.TriggerDelayPeriod);
Trig.ExternalTrigger(Settings.ExternalTrigger);
Trig.AtConfigure();

FBlockCount = 0;
FBlockRate = 0;
//
//  Check Channel Enables
int ActiveChannels = Module().Output().ActiveChannels();
if (!ActiveChannels)
    {
    Log("Error: Must enable at least one channel");
    UI->AfterStreamStop();
    return false;
    }
```

The trigger manager is an object defined in the Application portion of Malibu to handle how and when the application manages the software trigger during a run. Separating this kind of boilerplate code makes the ApplicationIo object cleaner, and also makes finding out how the examples use the trigger easier to discover as well.

In this example, if using a software trigger the trigger will be applied after the number of seconds given by the Trigger Delay period. This gives the streaming engine time to deliver substantial data to the card before activating the analog. If using an external trigger, the software trigger will not be applied.

```
    //
    // Trigger Configuration
    //  Frame Triggering
    Module().Output().Trigger().FramedMode((Settings.Framed)? true : false);
    Module().Output().Trigger().Edge((Settings.EdgeTrigger)? true : false);
    Module().Output().Trigger().FrameSize(Settings.FrameSize);
    //  Route External Trigger source
    IX6IoDevice::AfeExtSyncOptions syncsel[] = { IX6IoDevice::essFrontPanel, IX6IoDevice::essP16 };
    Module().Output().Trigger().ExternalSyncSource( syncsel[ Settings.ExtTriggerSrcSelection ] );
    //
    //  Velocia Packet Size
    Module.SetOutputPacketDataSize(Settings.PacketSize);
    //
    //  Output Test Generator Setup
    Module.SetTestConfiguration( Settings.TestGenEnable, Settings.TestGenMode );
    Module().Output().TestFrequency( Settings.TestFrequencyMHz * 1e6 );

    // Set Decimation Factor
    int factor = Settings.DecimationEnable ? Settings.DecimationFactor : 0;
    Module().Output().Decimation(factor);
```

This code sets up the trigger logic and routing, the packet size, decimation and the FPGA test generator. This last calls down into the ModuleIo object to handle any board-specific customization.

```
    //
    //  Configure Alert Enables
    Module.ConfigureAlerts(Settings.AlertEnable);
```

Alerts and starting the Stream are the same as in Input only mode. We call a board specific function to handle the details.

```
    // Disable prefill if in test mode
    Stream.PrefillPacketCount(Settings.TestGenEnable ? 0 : PrefillPacketCount);

    // Fill Waveform Buffer (if streaming)
    if (Settings.TestGenEnable == false)
        FillWaveformBuffer();
```

Like all of our Wave examples, streaming mode is handled by precalculating a wave pattern into a single buffer that is retransmitted each time data is required. The calculation of the buffer itself is a little more involved, but in the end there is a single Velo buffer that is repeatedly sent.

```
    Trig.AtStreamStart();

    //  Start Streaming
    Stream.Start();
    Log("Stream Mode started");

    return true;
}

//---------------------------------------------------------------------------
// ApplicationIo::StopStreaming()
//---------------------------------------------------------------------------

void ApplicationIo::StopStreaming()
{
    if (!IsStreaming())
        return;
```

```
    if (!FStreamConnected)
        {
        Log("Stream not connected! -- Open the boards");
        return;
        }

    //
    //  Stop Streaming
    Stream.Stop();

    Timer.Enabled(false);

    //  Disable test generator
    if (Settings.TestGenEnable)
        Module.SetTestConfiguration( false, Settings.TestGenMode );

    Trig.AtStreamStop();
}
```

The stream stop process remains simple. The trigger manager, periodic timer, and data stream objects are all stopped. In addition, the test generator is also turned off.

Data Required Event Handler

When the output stream needs additional data, the Data Required event is signalled. The Wave application uses this call to send the template block to the output via the SendOneBlock() method.

```
//---------------------------------------------------------------------------
// ApplicationIo::HandleDataRequired()
//---------------------------------------------------------------------------

void ApplicationIo::HandleDataRequired(PacketStreamDataEvent & Event)
{
    SendOneBlock(Event.Sender);
}

//---------------------------------------------------------------------------
// ApplicationIo::SendOneBlock()
//---------------------------------------------------------------------------
const int HeaderTagValuePostPacketizer = 0x00000000;
const int HeaderTagValueOriginal = HeaderTagValuePostPacketizer;

void ApplicationIo::SendOneBlock(PacketStream * PS)
{
    ShortDG Packet_DG(WaveformPacket);

    // Calculate transfer rate in kB/s
    double Period = Time.Differential();
    if (Period)
        FBlockRate = Packet_DG.SizeInBytes() / (Period*1.0e6);

    //
    //  No matter what channels are enabled, we have one packet type
    //    to send here
    PS->Send(0, WaveformPacket);

    ++FBlockCount;
}
```

For speed, the packet is created on the first call only. After that, the same data wave is sent to all channels. Note that it is allowed to send more than one output packet per notification. If no packets are sent, however, it is possible that further notifications may stop until the application starts sending data again. This decoupling of notification from sending allows different models of data generation to exist in Malibu. An application may send packets asynchronously and not handle notifications at all.

FillWaveformBuffer()
The Application section of the Malibu library has a waveform generator that will fill a packet with an interleaved set of channels. Sadly, this packet is not in the proper format for sending to the X6 cards, as it needs to be placed into the appropriate number of Vita packets with proper Stream Ids. These packets then need to be inserted into a Velocia buffer to send.

```
//---------------------------------------------------------------------------
//  ApplicationIo::FillWaveformBuffer() -- Fill buffer with waveform data
//---------------------------------------------------------------------------

void ApplicationIo::FillWaveformBuffer()
{
    //
    //  Builds a N channel buffer
    int channels = Module().Output().ActiveChannels();
    int bits = Module().Output().Info().Bits();
    int samples = static_cast<int>(Settings.PacketSize);

    // calculate scratch packet size in ints
    int scratch_pkt_size;
    if (bits <= 8)
        scratch_pkt_size = channels * Holding<char>(Settings.PacketSize);
    else if (bits <= 16)
        scratch_pkt_size = channels * Holding<short>(Settings.PacketSize);
    else
        scratch_pkt_size = channels * Settings.PacketSize;

    Innovative::Buffer  ScratchPacket;
    ScratchPacket.Resize(scratch_pkt_size);

    Builder.SampleRate(Settings.SampleRate*1e6);
    Builder.Format(channels, bits, samples);
    Builder.BuildWave(ScratchPacket);
```

The first step is to generate the actual wave data. The approach taken is to have the original wave generator construct a standard wave file in a scratch buffer, which we will then use to construct the correct Waveform packet. So we need to calculate the size of the scratch buffer and load the generator parameters. Then the BuildWave() call constructs the wave in the scratch buffer.

The second step is to copy the data from the scratch buffer into an array of Vita buffers. In this case we can do this as integers, as the data format is the same for our one stream ID. The maximum size of a Vita packet is relatively small, so we have to support paging into a number of them. We choose to make all the packets the same large size until the final one that holds the remainder.

```
    //
    //  We now have the data in a regular buffer. We need to copy it into
    //     VITA buffers, and then those VITAs into the Waveform packet
    std::vector<Innovative::VitaBuffer>  VitaQ;
    Innovative::UIntegerDG   ScratchDG(ScratchPacket);
    //
```

```
//  Bust up the scratch buffer into VITA packets
unsigned int words_remaining = ScratchPacket.SizeInInts();
unsigned int offset = 0;
while (words_remaining)
    {
    //  calculate size of VITA packet
    const unsigned int MaxVitaSize = 0xF000;   // Vita limited to 16 bit size

    unsigned int VP_size = std::min(MaxVitaSize, words_remaining);

    //  Get a properly cleared/inited Vita Header
    VitaBuffer VBuf = Stream.NativeVitaBuffer( VP_size );
    Innovative::UIntegerDG   VitaDG(VBuf);
```

The NativeVitaBuffer() function produces a buffer with a properly initialized header and trailer. This is particularly important here since the trailer's padding field, if not correct, will not allow you to completely fill the packet using our Datagrams.

What is padding? Padding is a response to the need to be able to issue a Vita packet at any time. To be send across the busmaster interface, the packet must have a size that is divisible by 4 (in 32 bit words). Yet we need to be able to issue a Vita packet in cases, such as when a trigger lowers, that can happen at any time. If we hold the data, it will be issued at the wrong time. To send it, we need to pad the packet to an even 128 bit boundary. The padding field is used to tell us which bytes in a packet are invalid pad bytes instead of actual data. The datagrams check this field and reduce the size of the datagram to avoid using this dead space for reading or writing.

```
    //  Copy Data to Vita
    for (unsigned int idx=0; idx<VitaDG.size(); idx++)
        VitaDG[idx] = ScratchDG[offset + idx];

    //  Init Vita Header
    VitaHeaderDatagram VitaH( VBuf );
    VitaH.StreamId( 0 );
    //  fix up loop counters
    words_remaining -= VP_size;
    offset += VP_size;

    //  save the buffer
    VitaQ.push_back( VBuf );
    }
```

Now that we have our array of Vita packets, we can copy them into a large Velo packet for sending. The VitaPacketPacker object was made to facilitate this operation. You can set an output packet size, and then as you Pack the packets in, whenever a new packet is filled, the OnDataAvailable() event is called for you to process the Velo packet before continuing. The expectation is that you would call Send() in this function, but in this case we will tweak the settings to make sure we end up with one buffer containing all the data.

The first step is to set the output size to be considerably larger than the total amount of data we have. Here I use a factor of 2. The real need is to have enough room for the packet data and the 8 words of header/trailer data for each packet. The rough doubling is good enough, since we only will have the one packet.

```
//
//  Use a packer to load full VITA packets into a velo packet
//  ...Make the packer output size so big, we will not fill it before finishing
VitaPacketPacker VPPk(scratch_pkt_size*2);
VPPk.OnDataAvailable.SetEvent(this, &ApplicationIo::HandlePackedDataAvailable);

//  ...Shove in our VITA packets
for (unsigned int i=0; i<VitaQ.size(); i++)
    {
```

```
            VPPk.Pack( VitaQ[i] );
            }

        VPPk.Flush();   // outputs the one waveform buffer into Waveform
```

At the end of the for loop above, all the data is present in the packer, with a large excess as well. The packer has the ability to force the output of residual data by truncating the packet to exactly fit. This Flush() operation forces a final call to the OnDataAvaliable event, which we have made sure is the only one we get.

Then we clean up and init the Waveform headers and we have a valid packet ready for sending when needed.

```
        // WaveformPacket is now correctly filled with data...

        ClearHeader(WaveformPacket);
        InitHeader(WaveformPacket);   // make sure header packet size is valid...

    }
```

This is the event handler for the Packer. In this case, we just do a simple assignment to save the data for later use. You can also add the buffer to a data structure, or Send() it at this time if the application is designed for that kind of I/O.

```
//---------------------------------------------------------------------------
//  ApplicationIo::HandlePackedDataAvailable() -- Packer Callback
//---------------------------------------------------------------------------

void  ApplicationIo::HandlePackedDataAvailable(Innovative::VitaPacketPackerDataAvailable & event)
{
    WaveformPacket = event.Data;
}
```

Developing Host Applications

Developing an application will more than likely involve using an integrated development environment (IDE) , also known as an integrated design environment or an integrated debugging environment. This is a type of computer software that assists computer programmers in developing software.

The following sections will aid in the initial set-up of these applications in describing what needs to be set in Project Options or Project Properties.

Borland Turbo C++

BCB10 (Borland Turbo C++) Project Settings

When creating a new application with File, New, VCL Forms Application - C++ Builder

Change the Project Options for the Compiler:

Project Options
++ Compiler (bcc32)
C++ Compatibility
Check 'zero-length empty base class (-Ve)'
Check 'zero-length empty class member functions (-Vx)'

In our example Host Applications, if not checked an access violation will occur when attempting to enter any event function.

i.e.
Access Violation OnLoadMsg.Execute – Load Message Event
Because of statement
Board->OnLoadMsg.SetEvent(this, &ApplicationIo::DoLoadMsg);

Change the Project Options for the Linker:

Project Options
Linker (ilink32)
Linking – **uncheck 'Use Dynamic RTL'**

In our example Host Applications, if not unchecked, this will cause the execution to fail before the Form is constructed.

Error: First chance exception at $xxxxxxxx. Exception class EAccessViolation with message "Access Violation!"
Process ???.exe (nnnn)

Other considerations:

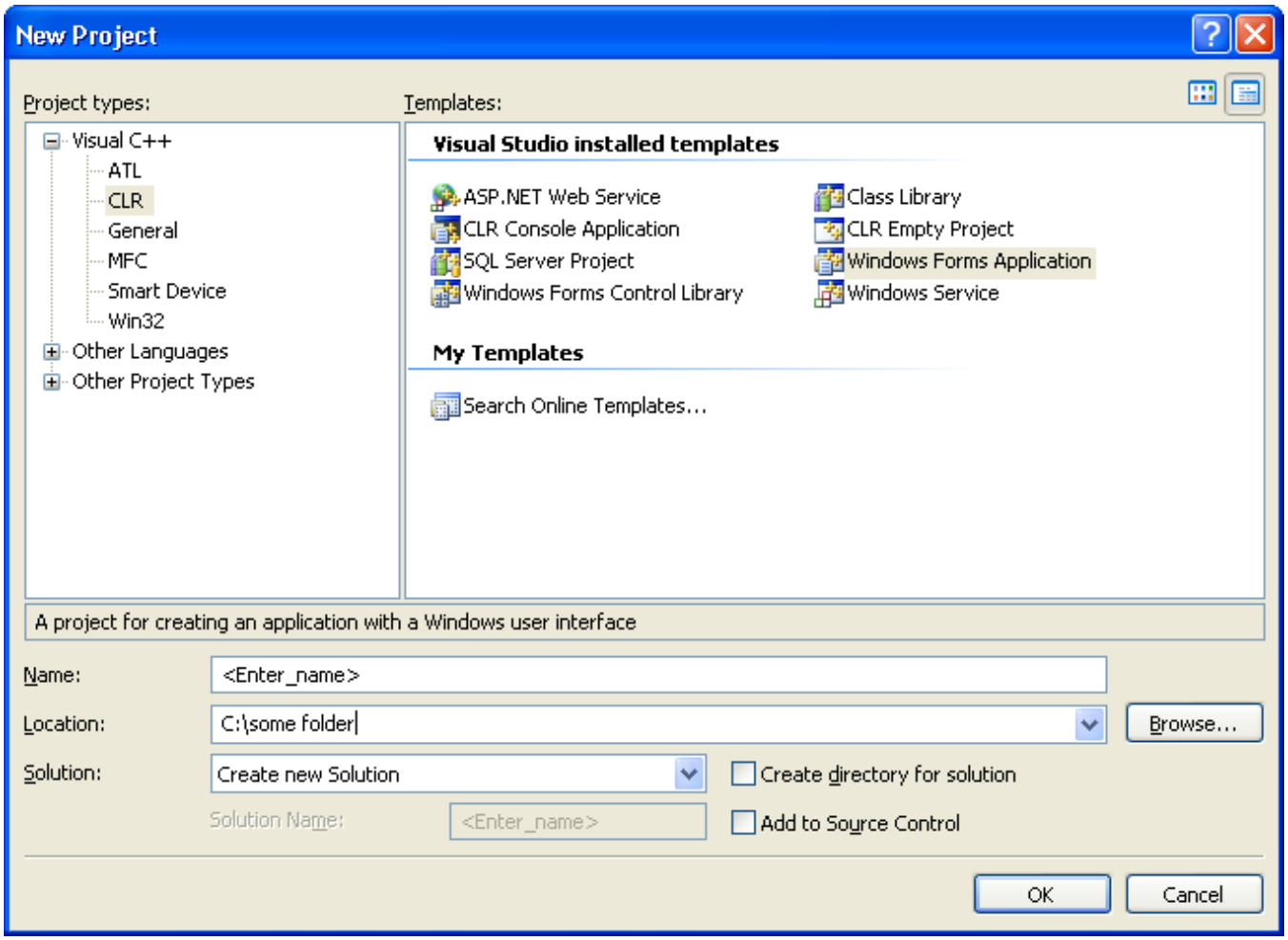
Project Options
++ Compiler (bcc32)
Output Settings
check – Specify output directory for object files(-n)
(release build) Release
(debug build) Debug
Paths and Defines
add Malibu
Pre-compiled headers
uncheck everything
Linker (ilink32)
Output Settings
check – Final output directory
(release build) Release
(debug build) Debug
Paths and Defines
(ensure that Build Configuration is set to All Configurations)
add Lib/Bcb10
(change Build Configuration to Release Build)
add lib\bcb10\release
(change Build Configuration to Debug Build)
add lib\bcb10\debug
(change Build Configuration back to All Configurations)

Packages
uncheck - Build with runtime packages

Microsoft Visual Studio 2005

Microsoft Visual C++ 2005 (version 8) Project Properties

When creating a new application with File, New, Project with Widows Forms Application:



Project Properties (Alt+F7)
Configuration Properties

Project Defaults	
Configuration Type	Application (.exe)
Use of MFC	Use Standard Windows Libraries
Use of ATL	Not Using ATL
Minimize CRT Use in ATL	No
Character Set	Use Unicode Character Set
Common Language Runtime support	Common Language Runtime Support (/clr)
Whole Program Optimization	No Whole Program Optimization

C++
General
Additional Include Directories

Malibu
PlotLab/Include – for graph/scope display

Code Generation
Run Time Library

Multi-threaded Debug DLL (/Mdd)

Precompiled Headers
Create/Use Precompile Headers

Not Using Precompiled Headers

Linker
Additional Library Directories

Innovative\Lib\Vc8

If anything appears to be missing, view any of the example sample code Vc8 projects.

DialogBlocks

DialogBLocks Project Settings (under Linux)

Project Options
[Configurations]
Compiler name = GCC
Build mode = Debug
Unicode mode = ANSI
Shared mode = Static
Modularity = Modular
GUI mode = GUI
Toolkit = <your choice wxX11, wxGTK+2, etc>
Runtime linking = Static or Dynamic, we use Static to facilitate execution of programs out of the box.
Use exceptions = Yes
Use ODBC = No
Use OpenGL = No
Use wx-config = Yes
Use insalled wxWidgets = Yes
Enable universal binaries = No
...
Debug flags = -ggdb -DLINUX
Library path = %INNOVATIVE%/Lib/Gcc/Debug, %WINDRIVER%/lib
Linker flags = %AUTO% -Wl, @%PROJECTDIR%/Example.lcf
IncludePath= -I%INNOVATIVE%/Malibu -I%INNOVATIVE%/Malibu/LinuxSupport %AUTO%

[Paths]
INNOVATIVE= /usr/Innovative
WINDRIVER= /usr/Innovative/WinDriver
WXWIN= /usr/wxWidgets-2.8-7 provided that this is the location where you have installed wxWidgets.

Summary

Developing Host and target applications utilizing Innovative DSP products is straightforward when armed with the appropriate development tools and information.

Chapter 5: *Vita Packet Format*

Overview

The X6 family of baseboards introduces a new form of data streaming that changes the format of the data packets streamed to and from the card. This chapter describes the format and the changes from previous use.

X6 Velocia Packets

Packet Header Format

Just as in the X5, data busmastered between the board and the application program is enclosed in a data packet. For the most part, this packet has not changed. What has changed is that the packet header is now 4 words long, rather than 2. Also, the total packet size now must be divisible by 4 words rather than 2 as well. This is a reflection of the new internal bus being made wider for increased efficiency.

Table 34. X6 Velocia (Velo) Packet Header

0	**Velocia Header Word**
1	(reserved)
2	(reserved)
3	(reserved)

As before, the first word in the header contains the information used to identify the packet and determine its size. The Peripheral ID is a tag value used to identify the data source – all packets with the same Peripheral ID are defined as making up a single stream of data. The size allows the parsers to find the next header.

Table 35. Velocia Header Word

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Peripheral ID								Packet size (in 32 bit words)																							

In the Malibu library there are additional classes to support these packets – VeloBuffer to hold packet data and VeloHeaderDatagram to access the header. The standard access datagrams will operate on VeloBuffers identically with the original buffers to access the data portion of the packet.

Packet Data Format

The contents of an X6 Velocia packet is not raw data as in earlier systems, but a stream of VITA-49 formatted packets that are processed individually. These packets can be split across Velo packets, so the start of a Velo packet does not mean that a Vita packet header follows. The initial data could be part of a previous Vita packet's data.

X6 Vita Packets

Like Velocia packets, Vita packets require information included with them in addition to the data to indicate the data source and other useful information. This header is seven words long. There is an additional trailer word for each packet that increases the total to 8 words. The total size of a Vita packet on the X6 must be a multiple of 4 words long.

The table below shows a minimal Vita packet – it contains only 4 words of data with its 8 words of header information.

Table 36. Vita Packet Format

0	**Header IF Word**
1	**Header SID Word**
2	**Header Class OUI Word**
3	**Header Class Info Word**
4	**Header Timestamp – Integer Seconds Word**
5	**Header Timestamp – Fractional Seconds High Word**
6	**Header Timestamp - Fractional-Seconds Low Word**
7	Packet Data 0
8	Packet Data 1
9	Packet Data 2
10	Packet Data 3

11	Trailer Word

The requirement of the Vita packet to remain aligned to a 4 word boundary conflicts with the need to dispatch a packet at once when events occur such as the trigger signal going off. If the packet remains unsent, part of the data will not arrive and even if it will eventually come in, part of the packet will have occurred at one time and part at potentially a much later time. To allow the packet to be sent in the absence of valid data, the concept of Padding was added.

A field in the trailer can be set to indicate to the destination which bytes in the last 16 bytes are not valid data. The application must ignore this data when processing the buffer. So in the case a trigger ends in the middle of a 4 word block, the packet can be padded to achieve alignment and the padding field in the trailer set to indicate this to the analysis routines.

The Access Datagram classes all recognize this padding value and reduce the size of the packet accordingly. This means the applications must initialize the Padding field before attempting to fill a packet.

Packet Header Format

The above table shows the arrangement of the seven header words in the header. The remainder of this section will describe the fields for each word

VITA Header IF word

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Packet Type				C	T	R	R	TSI		TSF		Packet Count				Packet Size															
0001				1	1	0	0																								

Packet type: set to IF Data packet with Stream Id, binary 0001.
C (bit 27): Optional Class field, enabled.

T (bit 26): Optional Trailer field, enabled.

TSI (bits 23-22): Timestamping format for integer seconds field, configurable.

Table 37. Timestamp Integer Seconds Options

TSI code	Meaning
00	No Integer-seconds Timestamp field included (Not supported)
01	UTC: seconds elapsed since January 1, 1970 GMT.
10	GPS: seconds elapsed since January 6, 1980 GMT.
11	Other: seconds elapsed since some documented starting time.

TSF (bits 21-20): Timestamping format for fractional seconds field, configurable between 01 (sample count) or 11 (free running).

Table 38. Timestamp Fractional Seconds Options

TSF code	Meaning
00	No Fractional-seconds Timestamp field included (Not supported)
01	Sample count timestamp: fractional seconds since last integer-seconds event, counting in samples.
10	Real time timestamp: counts in increments of 1 picosecond since last integer-seconds event. (Not supported)
11	Free running count timestamp: No relation to the integer-seconds field. Counts in samples.

Packet count: increments on each subpacket with the same Stream Id (PDN). It's allowed to roll over from 1111 to 0000.

Packet size: in 32-bit words, including the header; the packet size is always a multiple of 4 due to how the packets are handled internally.

VITA Header SID word

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Destination Mask																Stream ID															

The VITA-49 Stream Id field is split into two 16-bit fields: Destination Mask and Stream Id.

The Destination Mask will help route packets to their destinations (ie. PCIE, Aurora 0 or 1 etc.).

The Stream ID will be used much like the peripheral ID is for Velocia packets – to identify the source or meaning of the data in the packet.

VITA Header Class OUI Word
This word is reserved.

Vita Header Class Info Word
This word is reserved.

Vita Header Timestamp – Integer Seconds Word
This word contains the integer portion of the timestamp data.

Vita Header Timestamp – Fractional Seconds High and Low Words
These words contain the fractional portion of the timestamp data.

Vita Packet Trailer Format

VITA Trailer Word

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Enables								1111				State and Event indicators								Padding in bytes				E	Context Packet count						

State and Event Bits and Enable Bits

Enable bit position	State and event Indicator bit position	Indicator name
31	19	Calibrated Time
30	18	Valid Data
29	17	Reference lock
28	16	AGC/MGC Indicator
27	15	Detected Signal indicator
26	14	Spectral inversion
25	13	Over-range indicator
24	12	Sample Loss (over-run)

Setting the appropriate bit in the Enable field will cause the State/Event bit to follow the signals given in the table. These will flag if the enabled conditions occur in the current Vita packet.

Bits 20-23
These should always be set to 1

Context Packet Count
This number tells which context packet is associated with this packet. Context packets can give slowly changing information, such as temperature readings, into a data stream without burdening each data packet with data that is unlikely to be different from packet to packet.

Padding
The Padding field indicate how many padding bytes were added to align the current packet to a 4 word boundary. Padding values can be from 0 (no padding) to 15 (1 data byte valid in the last 4 words).

Table 39. Padding Example

Data n-3	Data n-4	Data n-5	Data n-6
Padding 1	**Data n**	**Data n-1**	**Data n-2**
Padding 5	Padding 4	Padding 3	Padding 2
Padding 9	Padding 8	Padding 7	Padding 6
Trailer (Padding Field Value = 9)			

The above table shows an example of padding. The logic had accumulated the data in grey when the trigger was disabled – seven bytes of data. The packet can not be sent because the data would not be properly aligned. The logic therefore adds the nine byte remainder to the packet as padding, and loads the value 9 into the padding field of the trailer.

Analysis routines that check the padding value will properly limit themselves to reading the valid data and ignore the padding bytes.

Table 40. Maximum Padding for X6 Boards

Source FIFO parallel samples	Event size (bits)	Maximum Padding (bytes)
1x16 (RX)	16	14
2x16 (400M)	32	12
4x8 (GSPS)	32	12
8x8 (GSPS DES)	64	8

The above table shows the maximum expected padding value for a particular X6 board. The RX, which has one 16 bit sample per packet for each period, could possibly have to pad 14 of 16 bytes in a 4 word block. Devices which send more data per sample period will have smaller maximum padding values.

Chapter 6: *Applets*

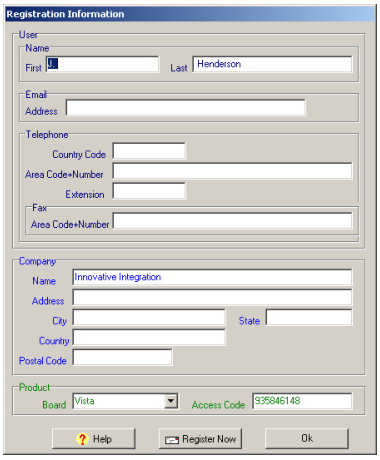
The software release for a baseboard contains programs in addition to the example projects. These are collectively called “applets”. They provide a variety of services ranging from post analysis of acquired data to loading programs and logic to a full replacement host user interface. The applets provided with this release are described in this chapter.

Shortcuts to these utilities are installed in Windows by the installation. To invoke any of these utilities, go to the Start Menu | Programs | <<Baseboard Name>> and double-click the shortcut for the program you are interested in running.

Common Applets

Registration Utility (NewUser.exe)

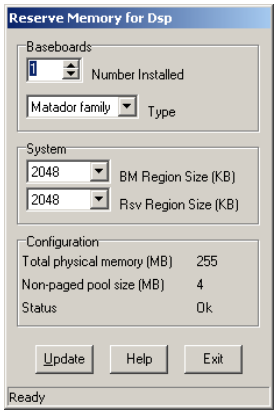
Some of the Host applets provided in the Developers Package are keyed to allow Innovative to obtain end-user contact information. These utilities allow unrestricted use during a 20 day trial period, after which you are required to register your package with Innovative. After, the trial period operation will be disallowed until the unlock code provided as part of the registration is entered into the applet. After using the NewUser.exe applet to provide Innovative Integration with your registration information, you will receive:



The unlock code necessary for unrestricted use of the Host applets

A WSC (tech-support service code) enabling free software maintenance downloads of development kit software and telephone technical hot line support for a one year period.

Reserve Memory Applet (ReserveMemDsp.exe)

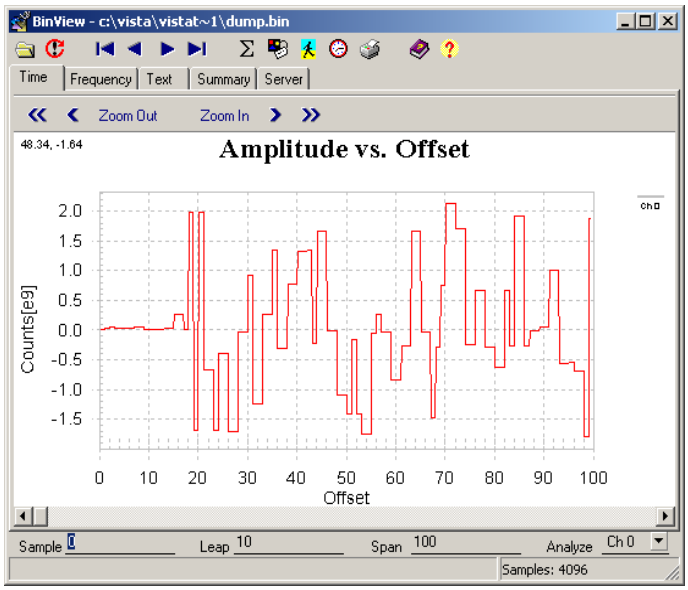


Each Innovative PCI-based DSP baseboard requires 2 to 8 MB of memory to be reserved for its use, depending on the rates of bus-master transfer traffic which each baseboard will generate. Applications operating at transfer rates in excess of 20 MB/sec should reserve additional, contiguous busmaster memory to ensure gap-free data acquisition.

To reserve this memory, the registry must be updated using the ReserveMemDsp applet. If at any time you change the number of or rearrange the baseboards in your system, then you must invoke this applet to reserve the proper space for the busmaster region. See the Help file ReserveMemDsp.hlp, for operational details.

Data Analysis Applets

Binary File Viewer Utility (BinView.exe)

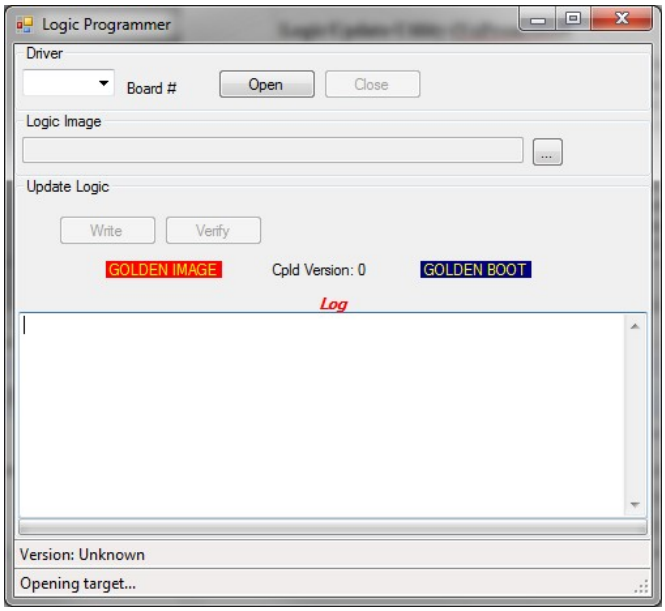


BinView is a data display tool specifically designed to allow simplified viewing of binary data stored in data files or a resident in shared DSP memory. Please see the on-line BinView help file in your Binview installation directory.

Chapter 7: *Applets for the X6 Family Baseboards*

Logic Update Utility (VsProm.exe)

The Logic Update Utility applet is designed to allow field-upgrades of the logic firmware on the X6 module. The utility permits an embedded firmware logic update file to reprogrammed into the baseboard Flash ROM, which stores the "personality" of the board.



Note that this utility should only be used after firmware development and debugging has been completed. During the development cycle, it is much more efficient to download and debug firmware using the Xilinx Bit-Blaster JTAG cable.

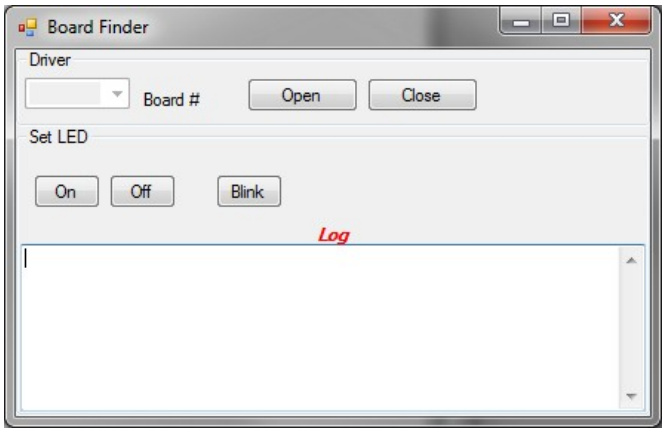
To use the applet, select the instance of the X6 module to be updated. This will be target zero in single-target installations.

Then, click the browse button (. . .) to select the logic bit file image containing the updated firmware image. Typically, this is located in the `Innovative\<Baseboard>\Hardware\Images` folder on your default drive.

Finally, click the Write button to program the firmware into the on-board FLASH rom. Programming typically takes about five minutes. After rebooting the PC, the new firmware will take effect.

Finder

The Finder is designed to help correlate board target numbers against PCI slot numbers in systems employing multiple boards.



Target Number

Select the Target number of the board you wish to identify using the Target Number combo box.

Blink

Click the Blink button to blink the LED on the board for the specified target. It will continue blinking until you click Stop.

On/OFF

Use the On and Off buttons to activate or deactivate (respectively) the LED on the baseboard for the specified target. When you exit the application, the board's LED will remain in the state programmed by this applet.

Chapter 8: *X6-RX XMC Module*

Introduction

The X6-RX is a member of the X6 XMC family that has four channels of 16-bit 160 MSPS A/D conversion with wide bandwidth analog inputs.

The X6-RX has a high performance computing core for signal processing, data buffering and system IO is built around a Xilinx Virtex-6 FPGA. Supporting peripherals include 1GBytes of LPDDR2 DRAM, conversion timebase and triggering circuitry, 64 bits of digital IO, Aurora or Serial Rapid IO secondary data port, and a host PCI or PCI Express interface. Features for module rugged operation including thermal and power monitoring support the computing and IO functions. The module format is a single slot XMC and is compatible with XMC.3 host sites.

Figure 31. X6-RX Module

Custom application logic development for the X6-RX is supported by the FrameWork Logic system from Innovative using VHDL and/or MATLAB Simulink. Signal processing, data analysis, and application-specific algorithms may be developed for use in the X6-RX logic and integrated with the hardware using the FrameWork Logic.

Software support for the module includes host integration support including device drivers, XMC control and data flow and support applets.

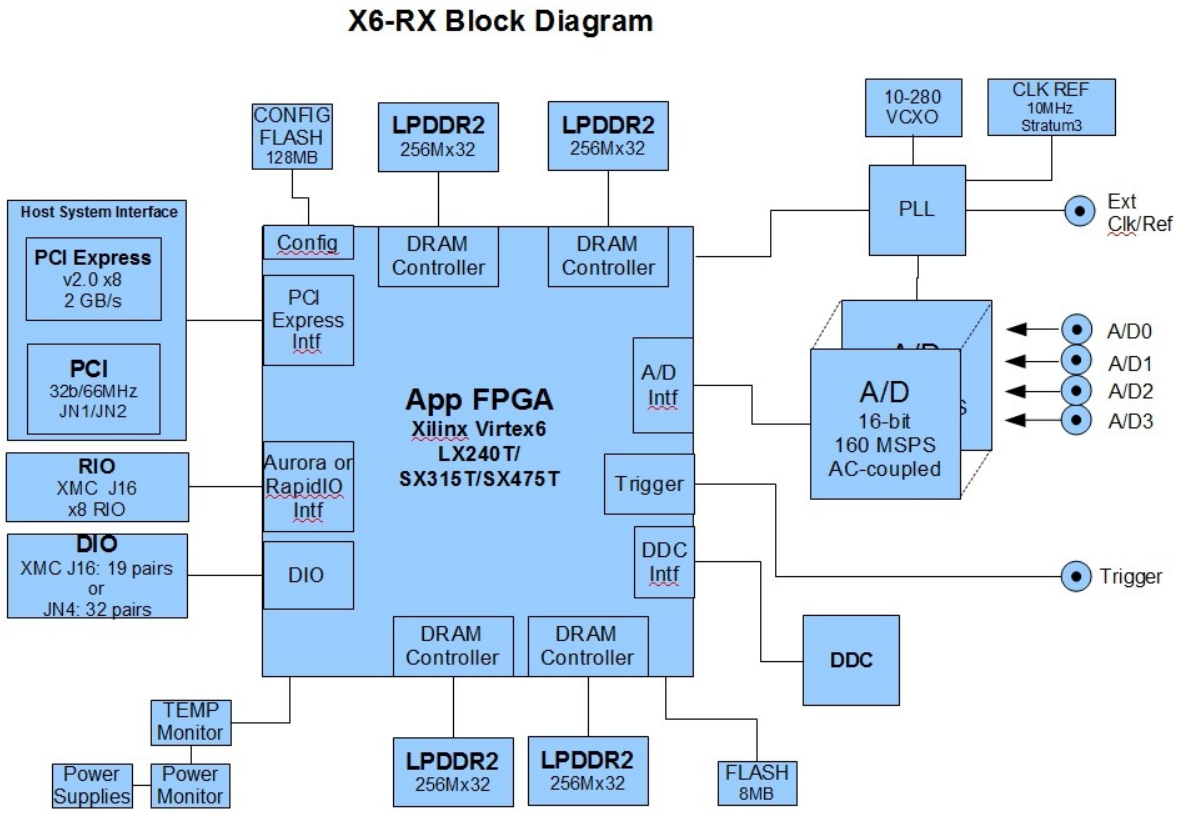


Figure 32. X6-RX Block Diagram

Hardware Features

A/D Converters

The X6-RX has four channels of 16-bit A/D sampling at up to 160 MSPS. The X6-RX has AC-coupled, 50 ohm terminated inputs through front panel SMA connectors.

Feature	Description
Inputs	4
Input Range	2.4Vpp
Maximum Input Current	30 mA
Input Coupling	AC-coupled
Input Impedance	50 ohm
A/D Devices	National Semiconductor ADC16DV160
Output Format	2's complement, 16-bit
Number of A/D Devices	4 simultaneously sampling
Sample Rate	1 MHz to 160 MHz
Calibration	Factory calibrated. Gain and offset errors are digitally corrected in logic. Non-volatile EEPROM coefficient memory.

Table 41. X6-RX A/D Features

Conversion clocking is provided by either a low jitter programmable clock source or an external clock input. The clock buffering is designed to minimize jitter and maximum acquired signal quality. See the clock discussion for more details.

A/D Front End

Front end circuitry for the A/D converters includes a 50 ohm terminated SMA input connector followed by a transformer into the A/D inputs. Impedance matching networks are used to provide the 50 ohm input.

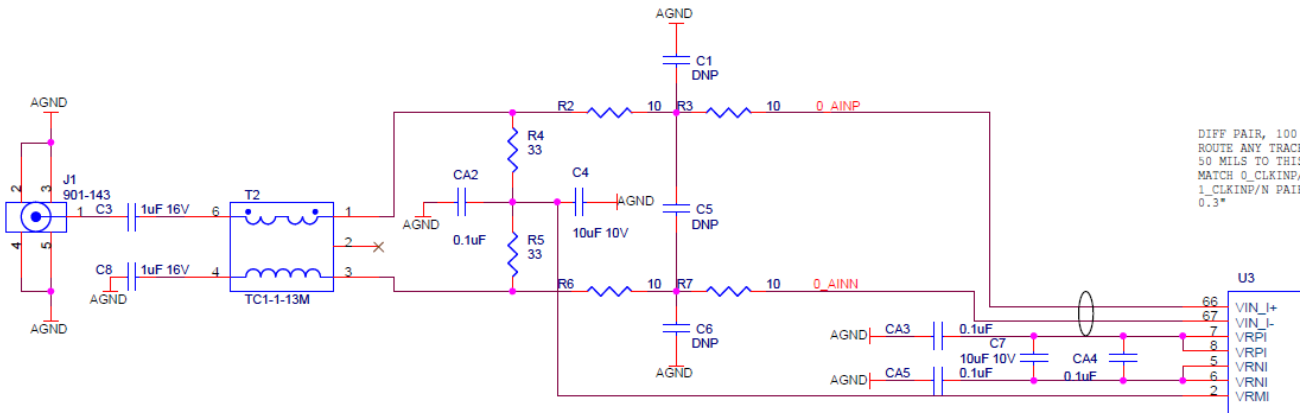


Figure 33. X6-RX A/D Channel Front End

The other 3 analog inputs are identical.

Input bandwidth measurement is shown in the data section of this chapter.

Input Range and Conversion Codes

Each high speed A/D input has a 2.4Vp-p single-ended input with 50 ohm input impedance. Other input ranges may be custom ordered.

Data output codes from the high speed A/D channels are 2's complement, 16-bits. The following table gives the transfer function.

Input voltage (V)	High Speed A/D Conversion Code (hex)
1.2	0x7FFF
0.6	0x3FFF
0	0x0000
-0.6	0xC000
-1.2	0x8000

Table 42. High Speed A/D Conversion Coding

Driving the A/D Inputs

The X6-RX has single-ended inputs that are 50 ohm terminated. The 50 ohm termination is used to match the input cable and connector characteristic impedance. The source signal must be able to drive this input impedance to achieve the best signal quality over the input voltage range. The signal source must be able to drive +/- 30mA for a full scale input. DC current limit is 30 mA.

Overrange Detection

The high speed A/D detects when an analog overrange occurs and generates an overrange flag. The logic receives this overrange detection and can trigger a system alert to notify the application when this error condition has occurred. The alert message shows when the overrange occurred in system time and which channels overranged.

Custom logic has access to the overrange bits in the A/D interface component. Each data sample indicates when an overrange occurs as part of its status byte appended to the data. This allows implementation of automatic gain controls for auto-ranging external front end signal conditioning.

Sample Rate Generation and Clocking Controls

Conversion clock sources on the X6-RX are on-card PLL or an external clock/reference input.

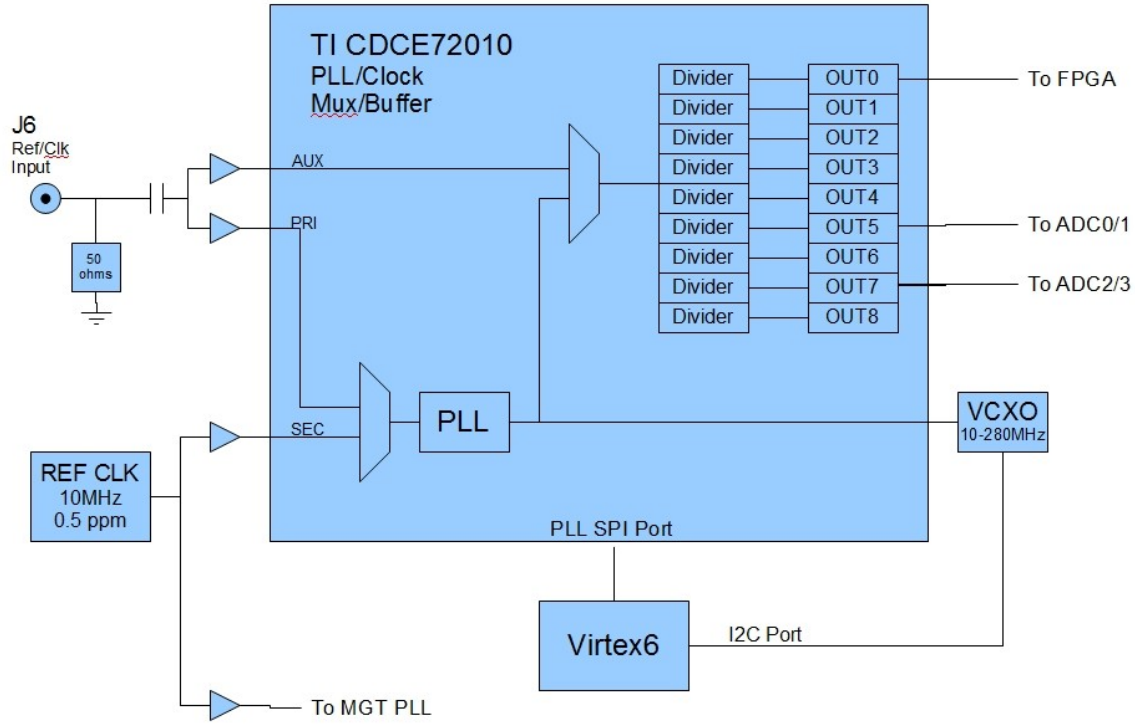


Figure 34. Sample Clock Generation and Distribution Block Diagram

Sample clocks are either generated by the PLL or are derived from an external input clock. Specialized clock circuitry is used on the X6-RX for clock generation and distribution since these clocks must be extremely low noise to achieve the best digitizing accuracy. This means that the clocks are NOT generated by the FPGA, but rather the specialized clock circuitry.

The clocks are copied to the FPGA with separate copies go to each A/D device. Programmable controls for the clock circuitry are mapped to the FPGA, accessed in the Framework Logic by the host.

Custom FPGA implementations can control the clocks without host involvement by commandeering these interfaces.

External Clock/Reference Input

The external clock/reference input at connector J6 is on the front panel and has the following electrical requirements for the clock input.

Characteristic	Description
Input Impedance	50 ohm
Input Coupling	AC
Input Connector	SMA
Minimum Input Amplitude	200mVp-p (-20.8 dBm)
Maximum Input Amplitude	2.0Vp-p (-0.8 dBm)
DC input range	+/-20V max
Maximum Frequency	500 MHz
Input waveform	Sine or square

Figure 35. Input Clock/Reference Electrical Specifications

This signal can be used as either a sample clock or as a PLL reference.

Sample Rate Generation

The PLL is used to generate sample clocks on the X6-RX using either an on-card programmable reference clock or an external reference input.

Parameter	Specification
PLL Clock Range	1 to 280 MHz
PLL Output Clock Resolution	100kHz
PLL Output Jitter	<200 fs RMS

Table 43. PLL Specifications

Setting the Sample Rate in the SNAP Example

The example software for the X6-RX illustrates the use of the sample clock controls and features.

The clock is selected internal (PLL) or external input. A frequency should be specified even when an external clock is used because the software uses this to estimate the data rate and size buffers appropriately. The external clock must be input on front panel connector J6.

The PLL reference is also selected to be internal (10 to 280 MHz reference) or external. For external reference the frequency must be exactly specified so that the PLL is programmed properly. External reference connector is J6 on the front panel.

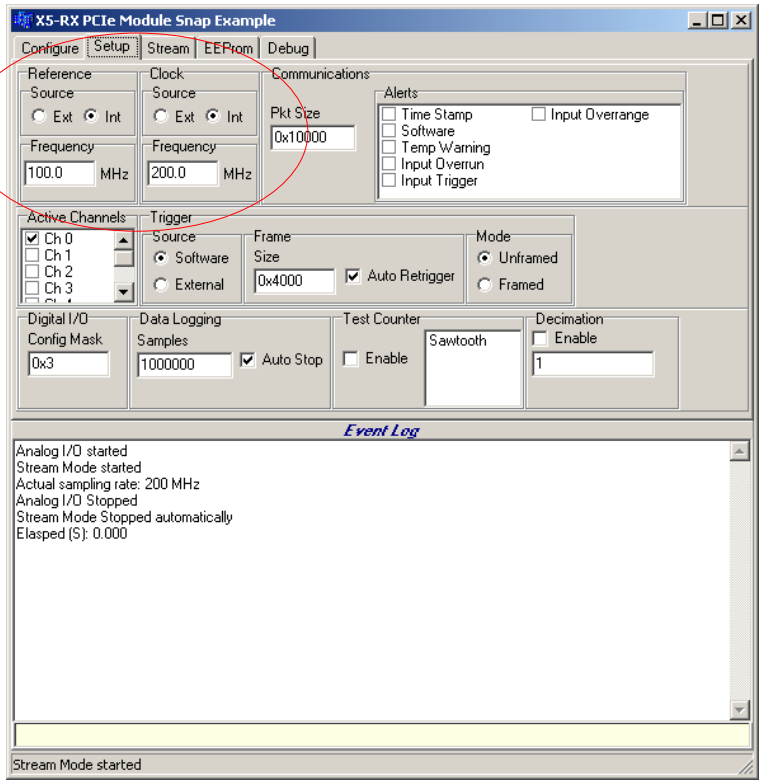


Figure 36. SNAP Example Sample Clock Controls

Controlling the PLL

How To Set the Sample Rate Generator to a Specific Frequency

To generate the settings for a desired sample rate, the PLL and VCXO are configured generate the closest possible frequency while meeting several restrictions.

PLL Parameter	Constraint
On-card PLL reference	10MHz with +/-5 ppm trim via I2C port

External Reference Input Range	1-500 MHz
VCXO Center Frequency Ranges	Programmable, 0.001 Hz resolution 10 to 280 MHz
Phase Comparator Set point	100 kHz

Table 44. Sample Rate Generation Parameters

Step	Calculation
1	Find an integer multiple, even numbers only, of the desired sample rate that is within the tuning range of the VCXO, where D = 1,2,4,6,8,10,12,14,16,18,20,24,28,32,64,80 $F_{VCO} = D*Fs$
2	Calculate the VCXO internal frequency to check operating mode. $F_{DCO} = F_{VCO} * HS\_DIV*N1$ select HS_DIV = 4,5,6,7,9 or 11 N1 = 1 to 128, even only so that $4850 < F_{DCO} < 5670$ MHz $HS\_DIV*N1 >= 6$ for $10 < F_{VCO} < 280$ MHz
3	Set the VCXO center frequency. $RFREQ = F_{DCO}/114.285$ MHz Use 38-bit math with 10 decimal places and 28 fractional bits.
4	Calculate PLL settings to generate the sample frequency Fs. The PLL analog loop filter is set for a phase comparator frequency of 1 MHz. Therefore, select R,P,B and A to satisfy these equations: $F_{REF}/R = 1$ MHz $F_{VCO}/(PB+A) = 1$ MHz where R= 1 to 16383 P= 1,2,2/3,4/5,8/9,16/17,32/33/ or 3 A= 0 to 63 Note: R=100 for on-card 100 MHz reference.

Table 45. Steps to Configure the VCXO and PLL

Driver code in the Malibu support libraries implements these calculation steps to program the PLL and VCXO. When these library functions are used, the software checks to verify that all restrictions for VCXO and PLL programming are met and that the output frequency is as close as possible to the desired result.

Using An External PLL Reference

The PLL can use an external clock input as its reference. This allows the sample clocks to be synchronous with the external clock. Many applications use this to synchronize multi-channel systems to sample simultaneously. Distributed applications can input time reference from GPS or other network time sources to synchronize systems.

The external clock must be low phase noise and stable to use as a PLL reference. Phase noise on the reference will directly result in phase noise on the generated clock. This means that the phase noise must be very low, typically less than 200 fS RMS, to be clean enough not to influence the acquired signal. The following graph shows the effect of jitter on the sample accuracy and noise level.

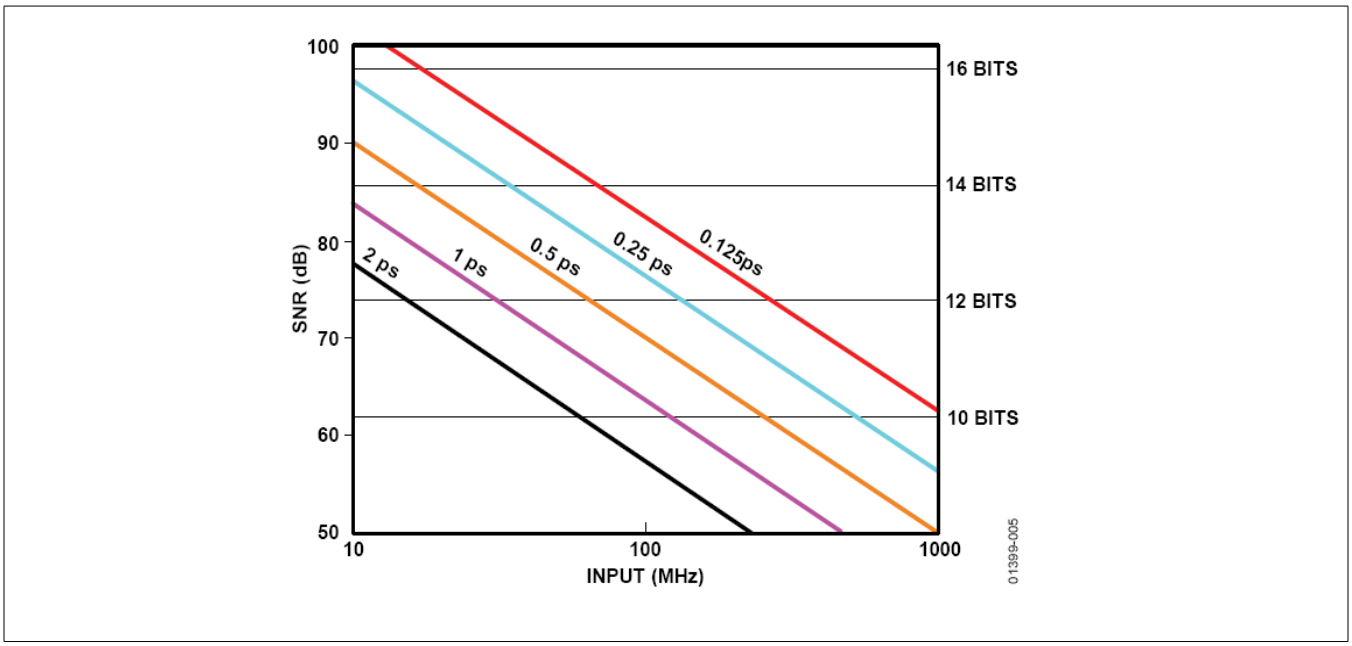


Table 46. Effect of Sample Clock Jitter on Digitizing Accuracy (Courtesy Analog Devices, Inc.)

The PLL reference clock multiplexer device also adds jitter to the input reference clock. This noise must be root-sum-squared (RSS) with the reference clock jitter.

Parameter	Worst
Additive Jitter	<100 fs in 10kHz to 20MHz range
Delay	450 ps (max)

Table 47. External PLL Reference Additive Jitter and Delay

The external clock must also be stable within the tracking range of the PLL/VCXO. This requirement limits the amount of low frequency wander and drift that external clock can have without making the PLL lose lock.

External Reference Clock Parameter	Limit
Frequency Stability	+/-3000 PPM
Jitter	200 fs RMS (recommended for analog input signals with < 500

	MHz bandwidth)

Table 48. External PLL Reference Requirements

The following diagram shows the clock path when an external reference is used. The reference clock multiplexer is configured to select the external clock input as the reference to the CDCE72010 device.

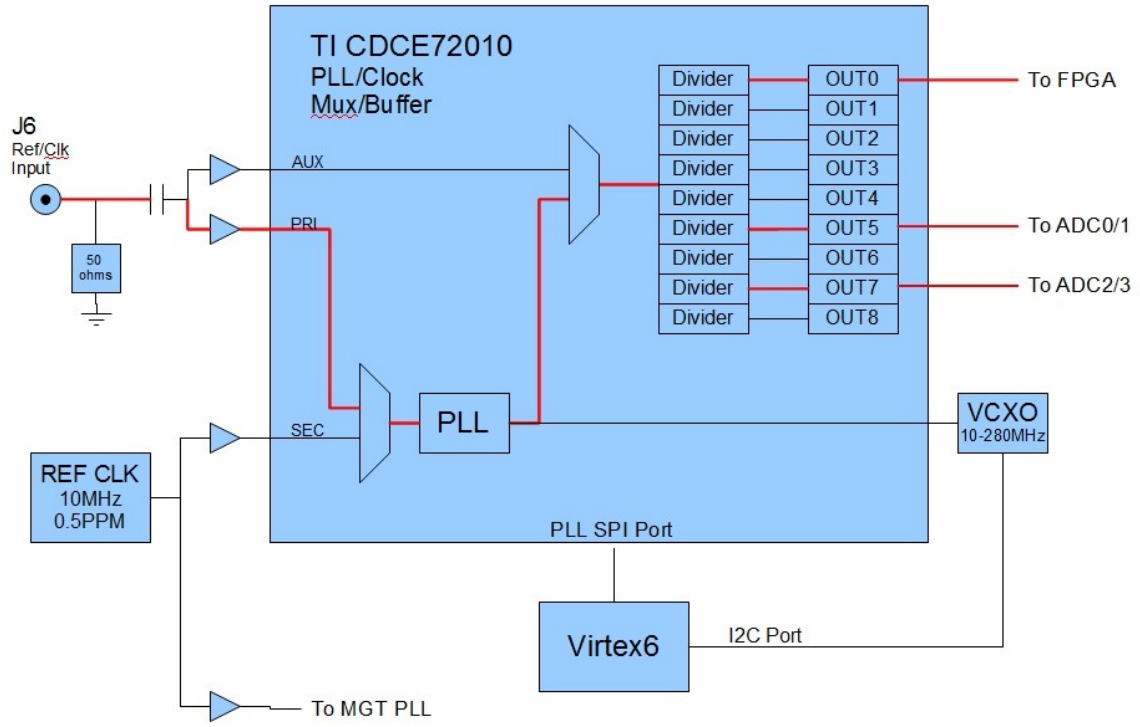


Figure 37. Clock Path Using External Reference Input

To use the external clock input as a reference, the CDCE72010 reference must be set to secondary input. This can be done using either a Malibu library function in software, from a script in the example programs, or set by the FPGA in custom logic.

Using An External Clock for Sample Clock

The external clock input on J6 (front panel) can be used as a sample clock. In this mode, the sample clock is buffered and distributed, with an option for clock division, to the DAC devices and FPGA.

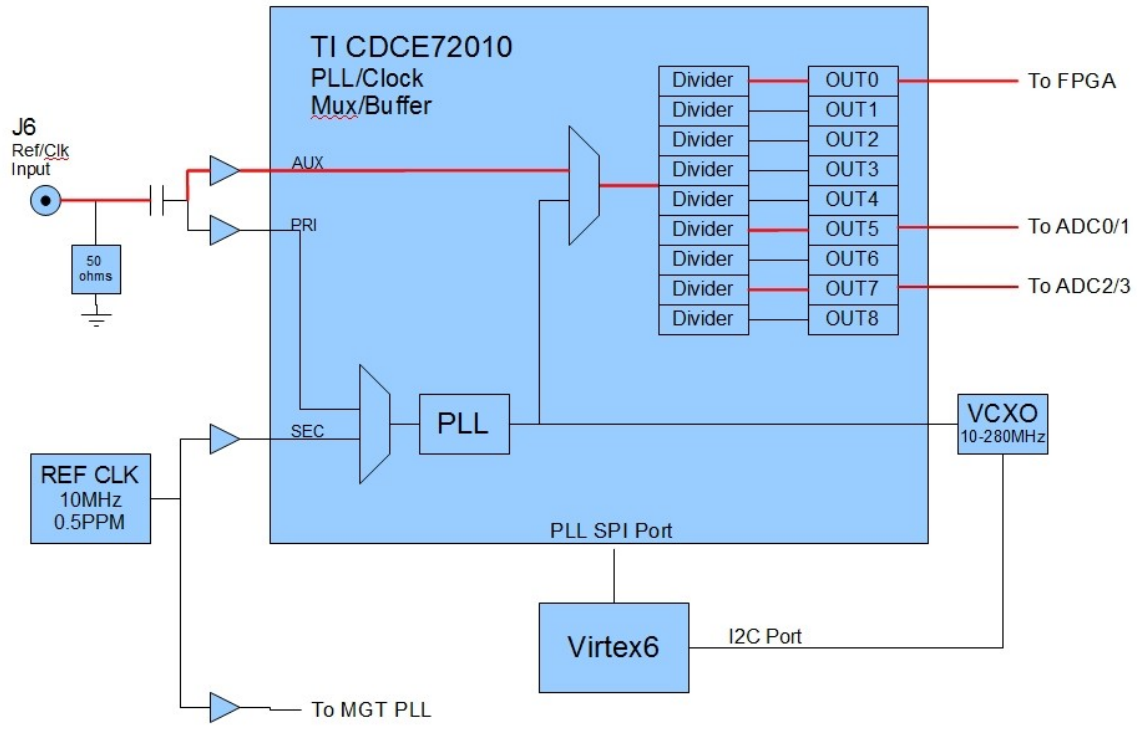


Figure 38. Clock Path Using External Clock Input

Configure the CDCDE72010 device to use the auxiliary clock input, then program the dividers for each output clock. These controls are mapped to the CDCDE72010 SPI port, mapped to the PCI Express bus in the Framework Logic. Custom logic implementations can control the PLL directly from the FPGA as well.

The Malibu libraries provide software functions for configuration of the CDCDE72010 device. This software configures the device for the clock selection and programs the output dividers.

External Clock Requirements

The external clock input has the following requirements. This signal is an AC-coupled input. Larger input amplitudes usually result in better A/D performance.

Parameter	Min	Typ	Max	Comments
Input Frequency	1 MHz		500 MHz	
Input Common Mode Input Voltage	-20V	0	+20V	
Input Amplitude	0.15 V		1.3 V	Peak-to-peak.
Input Termination		50 Ohms		
Input Capacitance		15 pF		Excludes cabling.

Table 49. External Clock Input Requirements

CDCDE72010 SPI Port

The CDCDE72010 PLL/Buffer device is configured through its SPI serial port. This port is mapped to the PCI Express bus as a memory mapped register at address BAR1 + 0x801. Writes to this register transmit to the CDCDE72010 device, reads from this address first transmit an address to the CDCDE72010 device then receive the current value. Before any read/write is performed, the SPI READY bit should be read to and must be true ('1').

Bits	Function
31..0	SPI write data

Table 50. PLL SPI interface – 0x801(r/w)

The CDCDE72010 has an extensive set of registers in the device for configuration and status. Consult the CDCDE72010 data sheet for details.

VCXO I2C Port

This register is an an I2C port that programs the VCXO for the PLL. The VCXO is a Silicon Labs SI571 device, offering a programmable center frequency controlled over its I2C port. Software functions in Malibu tools provide support for programming this device, including calculation of its register settings for use.

This I2C port is implemented using software controlled protocol. This means that the I2C port SDA and SCL connections must be controlled with the software to implement the I2C interface timing and commands. The control register is a simple pair of registers to control each signal and read back the pin. All device addressing, commands and data are read using the I2C software driver through the controls in this register.

Bit	Direction	Definition
0	W	SDA serial data bit
1	W	SCK bit for serial clock
2	R	Readback for I2C data pin
3	R	Readback for I2C clock pin
6..4	-	Unused
7	R/W	VCO output enable '0' = disabled (default)

31..8	-	Unused

Table 51. VCXO control and I2C register – 0x802 (r/w)

The I2C SDA (data) output is set using bit 0 and the pin is readback on bit 2. The I2C SCL (clock) is set using bit 1 and the pin readback is on bit 3. These signals must emulate the I2C pin functions for the protocol.

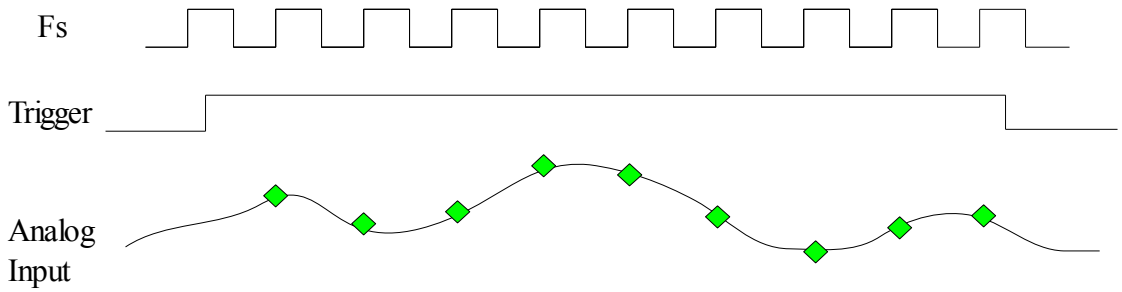
Triggering

The X6-RX has a trigger control component in the FPGA that controls the data acquisition process. The sample clock specifies the instant in time when data is sampled, whereas triggering specifies when data is kept. This allows the application to collect data at the desired rate, and keep only the data that is required.

On the X6-RX module, all A/D channels operate synchronously using the same clock and trigger. The trigger controls allow data to be acquired continuously, or during a specified time, as triggered by either a software or external trigger. Data can also be decimated to reduce data rates.

Trigger Mode	Data Collected/Played Back	Start Trigger	Stop Trigger
Continuous	All enabled channel pairs	Software or rising edge of external trigger	Software or falling edge of external trigger
Framed	N sample points for each of the enabled channel pairs	Software or rising edge of external trigger	Stops when N samples are collected back
Decimation	M points are discarded for every point kept. May be used with either trigger mode.	-	-

Table 52. Trigger Modes



Samples are acquired when trigger is true on rising edges of Fs when trigger is true.

Figure 39. Analog Triggering Timing

As shown in the diagram, samples are captured on the rising edges of the sample clock when the trigger is true. The trigger is true in continuous mode after a rising edge on the trigger input, software or external, until a falling edge is found. The trigger is timed against the sample clock and may have a 0 to +1 A/D conversion clock uncertainty for an asynchronous trigger input. To guarantee exact triggering no triggering uncertainty, the input trigger MUST be synchronous to the sample clock.

Trigger Sources

A software trigger or external trigger can be used by the trigger controls. Software trigger can always be used, but external triggering must be selected. The trigger source is level-sensitive for the continuous mode or edge-triggered for the framed mode triggering.

The Malibu software tools provide trigger source configuration and methods for software triggering, re-triggering in framed mode and trigger mode controls.

Trigger Source	Connector, Pin	Signal Standard	Notes
Front Panel	J5, center pin	AC-coupled DC input = 20V max Vin = 100 mV min, 1.5V max Termination = 50 ohms	
H_TRIG0	P16, pin A19	LVCMOS2.5 DC-coupled Vin = 3.6V max, 3V min	13.3K termination to ground.
H_TRIG1	P16, pin B19	LVCMOS2.5 DC-coupled Vin = 3.6V max, 3V min	13.3K termination to ground.
H_PPS	P16, pin D19	LVCMOS2.5 DC-coupled Vin = 3.6V max, 3V min	13.3K termination to ground.

Table 53. X6-RX Trigger Sources

Front Panel External Trigger Input Requirements

The front panel external trigger input has the following requirements. This signal is an AC coupled, single-ended input on connector J6.

Parameter	Min	Typ	Max	Comments
Input Frequency	1		200 MHz	
Input Amplitude	0.2 V		1.5V	
Input Termination		50		Ohms
Input Capacitance		15 pF		
DC Input Voltage			+/-10V	

Table 54. External Trigger Input Requirements

Framed Trigger Mode

Framed trigger mode is useful for collecting data sets of a fixed size each time the input trigger is fired. In framed mode, the trigger goes false once the programmed number of points N have been collected. Start triggers that occur during a frame trigger are ignored.

The maximum number of points per frame is 16,777,216 (2^24) points, while the minimum number of points is 8. Frame size must be a multiple of 8 on the X6-RX.

Data flow to the host is independent of the framed triggering mode. In most cases, packet sizes to the host are selected to be integer sub-multiples of the frame size to allow the entire data set to flow to the host. That way, the entire data frame can be moved immediately to the host without waiting for the next trigger frame.

Decimation

The data may be decimated by a programmed ratio to reduce the data rate. This mode is usually used when the data rate is less than the minimum master clock rate of the A/D. A/D performance is not specified and will degrade (sometimes with unpredictable results) if the converter is operated below its minimum sample rate.

The decimation simply discards M points for every point kept – no averaging or filtering is used. When decimation is true, the number of points captured in the framed mode is the number of decimated points, in other words the discarded points do not count. Maximum decimation rate is 1/4095.

When decimation is used in the framed trigger mode, the number of points captured is after decimation. The frame count is always the actual number of points inserted into the FIFO.

Trigger Controls in SNAP Example

The SNAP example application demonstrates the triggering modes and controls for the X6-RX. The trigger controls on the SETUP tab select the trigger source, mode and decimation.

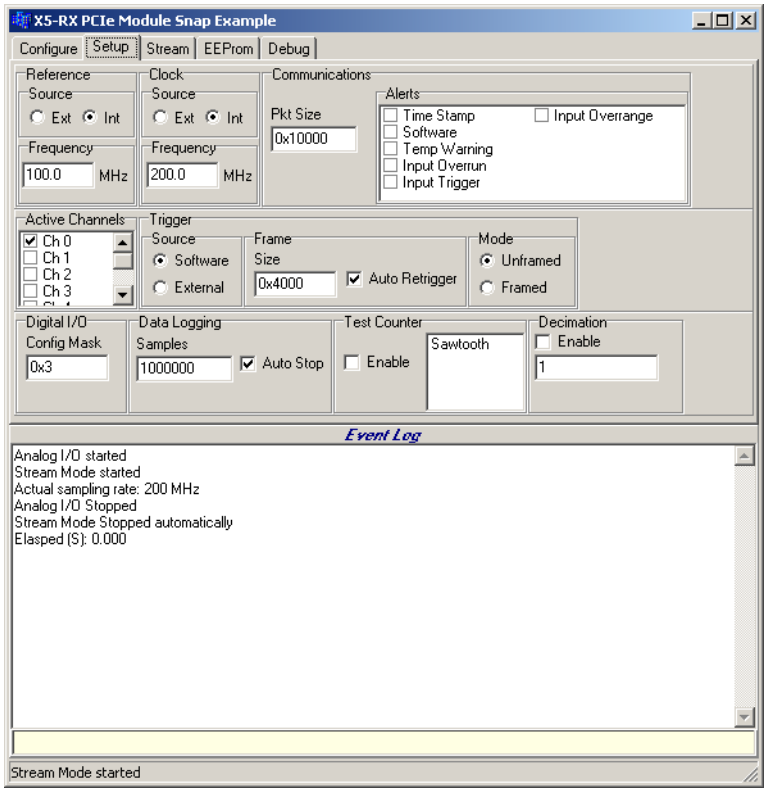


Figure 40. X6-RX SNAP Example Triggering Controls

The trigger source is either external or software. The front panel external trigger input is on connector J5. In external trigger mode, the data collection begins on rising edges. If you are in framed mode, then the number of points collected is set by the Frame Size number you enter. In unframed mode, the data is collected until the trigger is false.

One often misunderstood point about framed mode triggering is that it is best if the data packet size is set to the same, or integer sub-multiple packet size so that data will flow as expected. If the packet size is equal to the frame size, then the data packet will transfer to the system when the frame is complete. When this is mismatched, some or all of the data is "stuck" waiting for the packet to be completed. If multiple triggers are expected, then the next trigger may push the data out. If you have only one trigger, the data will not move to the host because the packet is not complete. It is best in most cases to just make the data frame that same as packet size so as to avoid this confusion.

Multi-A/D Synchronization

To synchronize multiple X6-RX cards, there are several requirements

- All cards must receive a synchronous clock or reference clock that is low jitter.
- All cards must receive a trigger signal that is a precisely time aligned to the input clock.

- All cards must be ready to take data when the trigger is fired. This means that all A/D have completed timing calibration and ready to take data.

Provided that these requirements are met, multiple cards can be synchronized for system expansion.

Digital Downconverter (DDC) ASIC

This feature is schedule for release in 7/2011.

DDC logic cores are available supporting wideband and narrowband downconversion for 4 to 256 channels. See Innovative IP catalog.

FrameWork Logic Functionality

The FrameWork Logic implements a data flow for the X6-RX that supports standard data acquisition functionality. This data flow, when used with the supporting software, allows the X6-RX to act as a data acquisition card with 1GB of data buffering and high speed data streaming to the host PCI/PCI Express. The example software for the X6-RX demonstrates data flow control, logic loading and data logging.

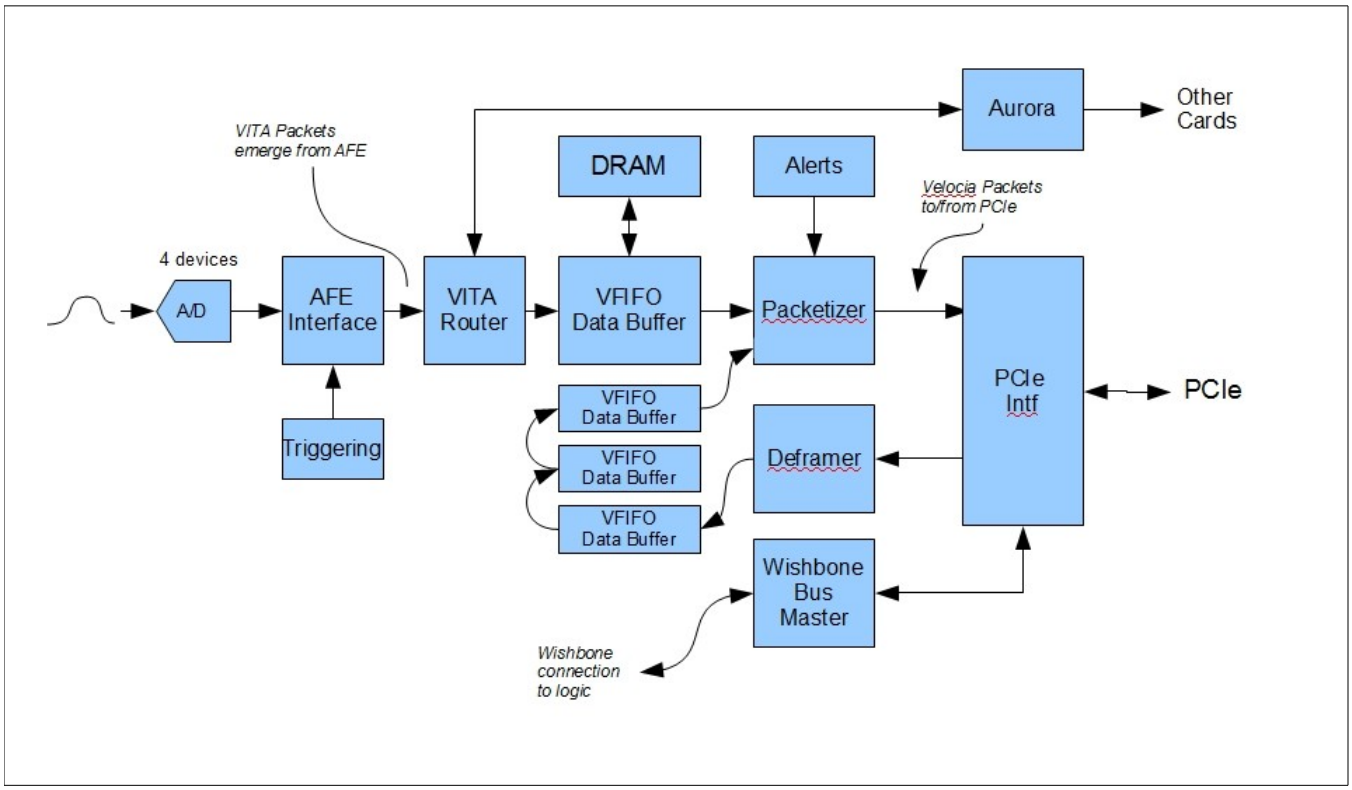


Figure 41. X6-RX FrameWork Logic Data Flow

The data flow is driven by the data acquisition process . Data flows from the A/D devices into the A/D interface component in the FPGA as controlled by the triggering. The data is then corrected for gain and offset errors associated with the analog inputs. After error correction, the enabled channels flow to the data buffer. The data buffer implements a data queue in the DRAM. The packetizer pulls data from the queue, creates data packets of the programmed size and sends those to the PCIe interface logic or out the host link. From here, the Velocia packet system controls the flow of data to the host. Data packets flow into host memory for consumption by the host program.

The Board Basics and Host Communications chapters of this manual discuss the use of the packet data system used on the X6 module family. The X6-RX module FrameWork Logic connects the data from A/D interface to the packet system by forming the data into 32-bit words of consecutive enabled channels. Status indicators for the A/Ds are integrated with the alert log to provide host notifications of important events for monitoring the data acquisition process, some of which are unique to the X6-RX.

The complete description of the FrameWork Logic is provided in the *X6-RX FrameWork Logic User Guide* including the memory mapping, register definitions and functional behavior. This logic is about 30% of the available logic in the application FPGA (Virtex6 LX240T/SX315T device). In many custom applications, unused logic functions can be deleted to free up gates for the new application.

Alert Log

Overview

X6 modules have an Alert Log that can be used to monitor the data acquisition process and other significant events. The application can use Alerts to create a time history of the data acquisition process that shows when important events occurred and mark the data stream to correlate system events to the data. This provides a precision timed log of all of the important events that occurred during the acquisition and playback for interpretation and correlation to other system-level events. Alerts for critical system events such as triggering, data overruns, analog overrange, and thermal warnings provide the host system with information to manage the module.

The Alert Log creates an alert packet whenever an enabled alert is active. The packet includes information on the alert, when it occurred in system time, and other status information. The system time is kept in the logic using a 32-bit counter running at the sample clock rate. Each alert packet is transmitted in the packet stream to the host , marked with a Peripheral Device Number corresponding to the Alert Log.

The Alter Log allows X6 modules to provide the host system with time-critical information about the data acquisition to allow better system performance. System events, such as over-ranges, can be acted on in real-time to improve the data acquisition quality. Monitoring functions can be created in custom logic that triggers only when the digitized data shows that something interesting happened. Alerts make this type of application easier for the host to implement since they don't require host activity until the event occurs.

Types of Alerts

Alerts can be broadly categorized into system, IO and software alerts.

System alerts include monitoring functions such as temperature, time stamp rollover and PLL lost. These alerts just monitor that the system is working properly. The temperature warning should be used increase temperature monitor and to prepare to shut down if necessary because thermal overload may be coming. Better to shut down than crash in most cases. The temperature failure alert tells the system that the module actually shut itself down. This usually requires that the module be restarted when conditions permit.

The data acquisition alerts, including over ranges, overflows and triggering, tell the system that important events occurred in the data acquisition process. Overflow is particularly bad – data was lost and the system should try to alleviate the problem by unclogging the data pipe, or just start over. If you get an overrange alert, then the data may just be bad for a while but acquisition can continue. Modules with programmable input ranges can use this to trigger software range changes.

Software alerts are used to tag the data. Any message can be made into an alert packet so that the data stream logged includes system information that is time-correlated to the data.

Table 55. Alert Types

Alert	Purpose
Timestamp rollover	The 32-bit timestamp counter rolled over. This can be used to extend the timestamp counter in software.
Software Alert	The host software can create alerts to tag the data stream.
ADC Queue Overflow	The ADC data queue overflowed indicating the the host did not consume

	the data quickly enough.
ADC Trigger	The ADC trigger went active.
ADC Overrange	An ADC channel was overranged

Alert Packet Format

Alert data packets have a fixed format in the system The Peripheral Device Number (PDN) is programmable in the software and is included in the packet header, thus identifying the alert data packets in the data stream. The packet shows the timestamp in system time, what alerts were signaled and a status word for each alert.

Dword #	Description	Contents
0	Header 1:	PDN & Total #, N, of Dwords in packet (e.g. Headers + data payload)
1	Header 2:	0x00000000
2	Alerts Signaled	Alerts(31..0)
3	Timestamp	(x"1303000" & "000" & timestamp_rollover)
4	0	0x00000000
5	Software Word	Software Word
6	Temperature sensor	(x"1303000" & "000" & temp_alert)
20..7	0	0x00000000
29	Memory status	(x"1303" & mem_alert_dout) mem_alert_din <= (vfifo3_afull & vfifo3_aempty & vfifo3_underflow & vfifo3_overflow & vfifo2_afull & vfifo2_aempty & vfifo2_underflow & vfifo2_overflow & vfifo1_afull & vfifo1_aempty & vfifo1_underflow & vfifo1_overflow & vfifo0_afull & vfifo0_aempty & vfifo0_underflow & vfifo0_overflow);
35..30	0	0x00000000

Table 56. Alert Packet Format

Since alert packets contain status words such as temperature for each packet, a software alert can essentially be used to read temperature of the module and so that it can be recorded.

Software Support for Alerts

Applications have different needs for alert processing. Aside from the bulk movement of data, most applications require some means of handling special conditions such as post-processing upon receipt of a stop trigger or closing a driver when an acquisition is completed.

When the alert system is enabled, the module logic continuously monitors the status of the peripheral (usually analog) hardware present on the baseboard and generates an alert whenever an alert condition is detected. It's also possible for application software to generate custom alert messages to tag the data stream with system information.

The Malibu software provides support for alert configuration and alert packet processing. See the software manual for usage.

Tagging the Data Stream

The Alert Log can be used to tag the data stream with system information by using software alerts. This helps to provide system-level correlation of events by creating alert packets in the data stream created by the host software. Alert packets are then created by the X6 module and are in the stream of data packets from the module. For example it is often interesting when something happens to the unit under test, such as a change in engine speed or completion of test stimulus.

Calibration

Each X6-RX is calibrated as part of the production testing. The calibration results are provided on the production test report with each module. The results of the calibration are stored in the on-board EEPROM memory. These calibration values are used by the logic to correct the analog errors for the A/D channels. Each input has a offset and gain calibration coefficients that are loaded by the software at initialization.

All factory calibrations limit gain correction to +-5% and offset to +/-2 mV.

All test voltages are measured as part of the procedure with NIST traceable equipment. Production calibration is performed at room temperature (~24C) with the module operating temperature at about 55C.

Under normal circumstances, calibration is accurate for one year. For recalibration, the module can be sent to Innovative or re-calibrated using a similar test procedure.

Updating the Calibration Coefficients

A software applet for writing the calibration coefficients to the EEPROM is provided (EEPROM.exe). New coefficients are simply typed into the offset and gain field for each channel.

Calibration coefficients for gain should not be outside the range of 0.95 to 1.05, and offset should not be outside the range of +/- 1000 counts for the A/Ds . If the calculated coefficients are larger than this, they are either wrong or the channel is damaged.

Using the X6-RX

Where to start?

The best place to start with the X6-RX module is to install the module and use the SNAP example to acquire some data. This program lets you log data from the module and use all the features like triggering, clocks, alerts and calibration ROM. You can use this program to acquire some data and log it to disk. This should let you verify that the module can acquire the data you want and give you a quick start on deciding what sample rates to use, how to trigger the data acquisition best for your application, and just get familiar with using the module.

The program also shows how to use BinView, a data analysis and viewing program by Innovative, that will let you see what you acquired in detail. Both time domain and frequency domain data can be viewed and analyzed. Data can also be exported to programs like Excel and MATLAB for further analysis.

Before you begin to write software, taking a look at SNAP will allow you see everything working. You can then look at the code for SNAP and modify it for your application or grab code from it that is useful.

Getting Good Analog Performance

The X6-RX is capable of digitizing very high frequency signals. To maximize signal to noise ratio and spur performance, it is important to use do the following

- Use only low jitter clock sources. The higher the input/output frequency, the more sensitive the system will be to clock jitter.
- Band limit input signals if possible. This avoids noise contributions and aliasing caused by out-of band energy in the input signal
- Scale your input signals to take advantage of the full scale input ad output ranges of the converters. This will maximize signal to noise in most cases. Custom input and output ranges can be ordered if necessary.
- Use high coax cables at all times, and terminate all signals to 50 ohms. Cables should be RG-179 or better.
- Reference input signals to the module ground. Be sure not to introduce ground loops.
- Provide sufficient signal strength to drive the input. The X6-RX terminates its inputs to 50 ohms, so signals sources must be able to drive the DC termination effectively.

If you decide to test the X6-RX to verify its performance, be aware that most signal sources are not good enough without additional filtering and careful use. Most single-ended lab instruments are limited by their distortion, especially at higher

frequencies, to about 70 dB. It is usually necessary to bandpass filter lab signal sources to improve the quality of the test signal before the X6-RX input. The filter reduces out-of-band noise and harmonic distortion from the signal source.

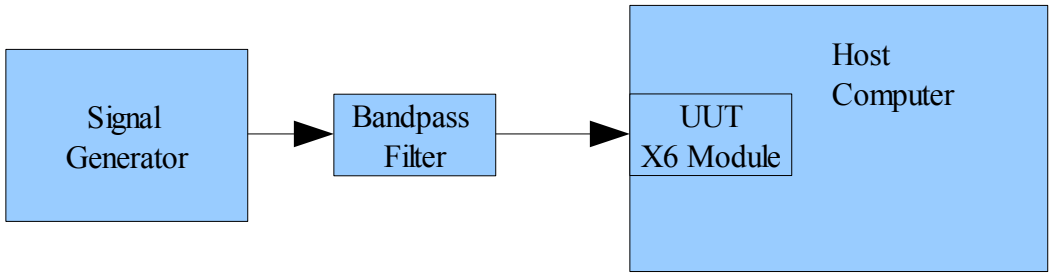


Figure 42. Typical Performance Evaluation Setup

Power Requirements

Power Supplies

The X6-RX requires the following voltages and power.

Supply	Voltage Range (V)	Nominal Voltage (V)	Maximum Allowed Current (A)	Allowable Noise	Derived from	Supplies these Devices
3.3V	3.125 to 3.465	3.3V	15	20 mV	Direct connect to the PCIe host through P15	FPGA, clock controls, and analog power supplies
VPWR	4.5 to 14V	12V	4	100 mV	Direct connect to the PCIe host through P16 (VPWR pins)	FPGA

Table 57. X6-RX Power Supply Requirements

The allowable noise allows the module to function, but may not give the best performance. Spectral content of the noise may affect analog performance .

Power Consumption

The X6-RX requires the following power for typical operation with when using the FrameWork Logic. This typical number assumes a 200 MHz system clock rate and 160 MSPS A/D sample rates for the application logic.

Voltage	Maximum Allowed Current (A)	Typical Current Required (A)	Typical Power (W)
3.3V	15	1.25	4.1
VPWR =12V	4	0.85	10.2
Total Power			**14.2****

Table 58. X6-RX Baseline Power Consumption

**This power consumption is the 'baseline power". The baseline configuration uses a modified version of the FrameWork Logic that has all 4 A/D channels sampling at 200 MSPS and streaming data to the host PCIe using x4 lanes. There are no Aurora ports, 3 DRAM devices are powered down, unused clocks are powered down. This baseline power consumption is the minimum functionality to use the card as a data acquisition card. Enabling other features will consume more power.

Feature	Voltage Supply	Power Consumption (W)	Notes
Aurora Ports	VPWR	0.13W per port	GTX at 2.5 Gbps, no load
A/D	VPWR	1.1W per A/D device (2 channels)	Power enables are provided to each A/D device with software control.
LPDDR2	VPRW	0.4W per device	3 devices are not used for baseline functionality. Software controls are provided for power down.

Table 59. X6-RX Power Consumption for IO Features

It has also been shown that running VPWR at 5V saves power, about 600 mW.

Surge currents occur initially at power-on and after application logic initialization. The power-on surge current lasts for about 10 ms at several amperes on both 3.3V and 12V. This surge is due primarily to charging the on-card capacitors and the startup current of the FPGAs. After initial power-up, the logic configuration will also result in a step change to the current consumption because the logic will begin to operate. In our testing and measurements, this has not been a surge current as much as a just a step change in the power consumption.

Power consumption varies and is primarily as a function of the logic design. Logic designs with high utilization and fast clock rates require higher power. Since calculating power consumption in the logic requires many details to be considered, Xilinx tools such as XPower are used to get the best estimates.

It is important that any custom logic design have a substantial safety margin for the power supply. Allowance for decreased power supply efficiency due to heating can account for 10% derating. Also, dynamic loads should be considered so that peak power is adequate. In many cases a factor of 2 for derating is recommended.

Environmental Ratings

The X6-RX is available for environmental rating levels from L0 (office, lab environment) to L4 (military and heavy industry).

Environment Rating <ER>		**L0**	**L1**	**L2**	**L3**	**L4**
Environment		Office, controlled lab	Outdoor, stationary	Industrial	Vehicles	Military and heavy industry
Applications		Lab instruments, research	Outdoor monitoring and controls	Industrial applications with moderate vibration	Manned vehicles	Unmanned vehicles, missiles, oil and gas exploration
Cooling		Forced Air 2 CFM	Forced Air 2 CFM	Conduction	Conduction	Conduction
Operating Temperature		0 to +50C	-40 to +85C	-20 to +65C	-40 to +70C	-40 to +85C
Storage Temperature		-20 to +90C	-40 to +100C	-40 to +100C	-40 to +100C	-50 to +100C
Vibration	Sine	-	-	2g 20-500 Hz	5g 20-2000 Hz	10g 20-2000 Hz
	Random	-	-	0.04 g^2/Hz 20-2000 Hz	0.1 g^2/Hz 20-2000 Hz	0.1 g^2/Hz 20-2000 Hz
Shock		-	-	20g, 11 ms	30g, 11 ms	40g, 11 ms
Humidity		0 to 95%, non-condensing	0 to 100%	0 to 100%	0 to 100%	0 to 100%
Conformal coating			Conformal coating	Conformal coating, extended temperature range devices	Conformal coating, extended temperature range devices, Thermal conduction assembly	Conformal coating, extended temperature range devices, Thermal conduction assembly, Epoxy bonding for devices
Testing		Functional, Temperature cycling	Functional, Temperature cycling, Wide temperature testing	Functional, Temperature cycling, Wide temperature testing Vibration, Shock	Functional, Temperature cycling, Wide temperature testing Vibration, Shock	Functional, Testing per MIL-STD-810G for vibration, shock, temperature, humidity

Table 60. X6-RX Environmental Limits

Testing for each unit is performed to verify compliance with the specified requirements. For levels above L0, functional testing is performed over the specified temperature range and functional testing is verified during vibration tests. Shock testing is non-operation, with post-shock functional testing.

Performance Data

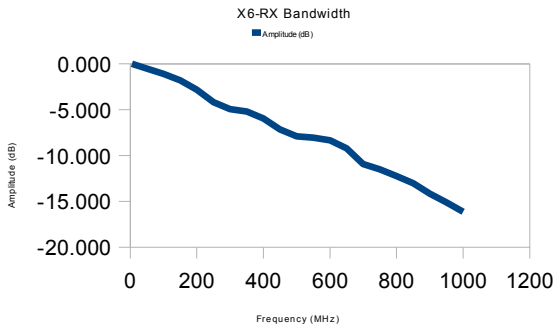
Analog Input Performance Summary

A summary of the analog performance follows for the X6-RX module.

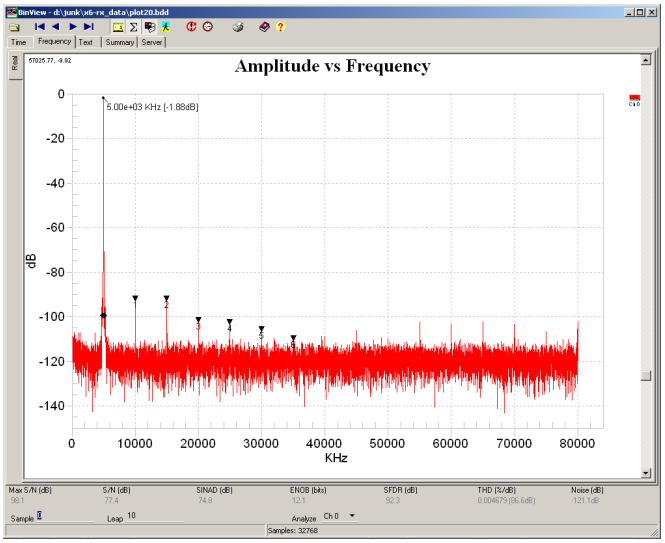
All tests performed at room temperature, with no forced air cooling unless noted. Test environment was PCIe adapter card in PC running testbed software using FrameWork Logic.

Table 61. X6-RX Analog Performance Summary

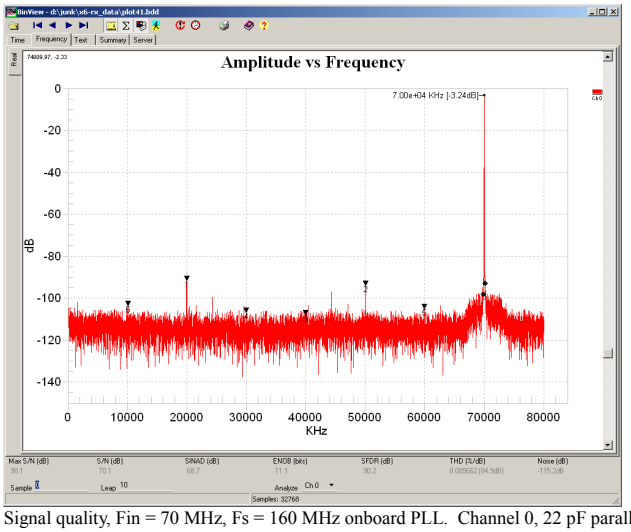
Parameter	Typ	Units	Notes
Analog Input Bandwidth	200	MHz	-3dB, 22 pF filter cap at A/D device inputs
SFDR	86	dB	70 MHz sine input, 85%FS, Fs = 160 MSPS
S/N	68	dB	70 MHz sine input, 85%FS, Fs = 160 MSPS
THD	0.01	%	70 MHz sine input, 85%FS, Fs = 160 MSPS
ENOB	11	bits	70 MHz sine input, 85%FS, Fs = 160 MSPS
Channel Crosstalk	<80	dB	70 MHz, 2Vp-p on adjacent channels
Noise Floor	-110	dB	Input Grounded, Fs = 160 MSPS, 32K sample FFT, non-averaged
Gain Error	<0.2	% of FS	Calibrated
Offset Error	<500	µV	Calibrated



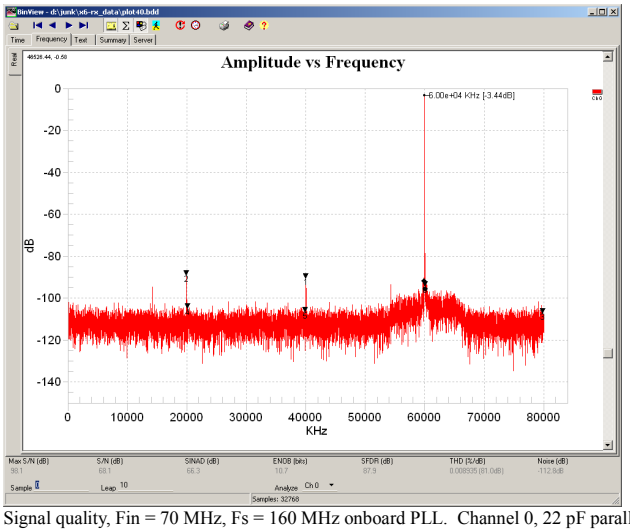
X6-RX Input Bandwidth (22 pF parallel cap at A/D device input)



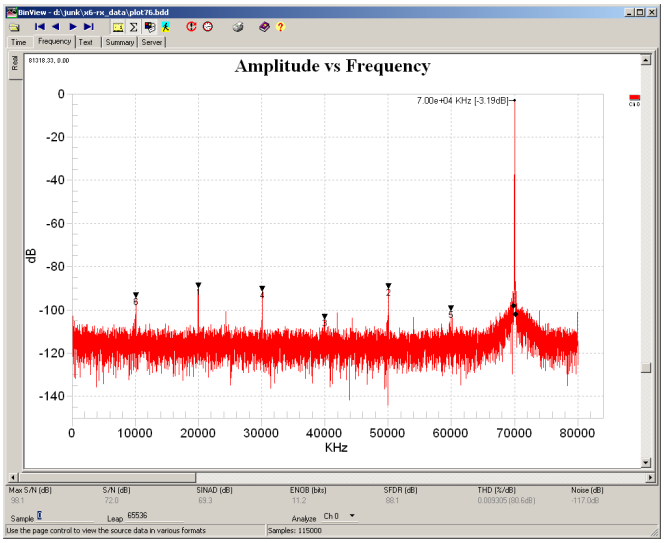
Signal quality, Fin = 5 MHz, Fs = 160 MHz onboard PLL. Channel 0, 22 pF parallel cap at A/D device inputs



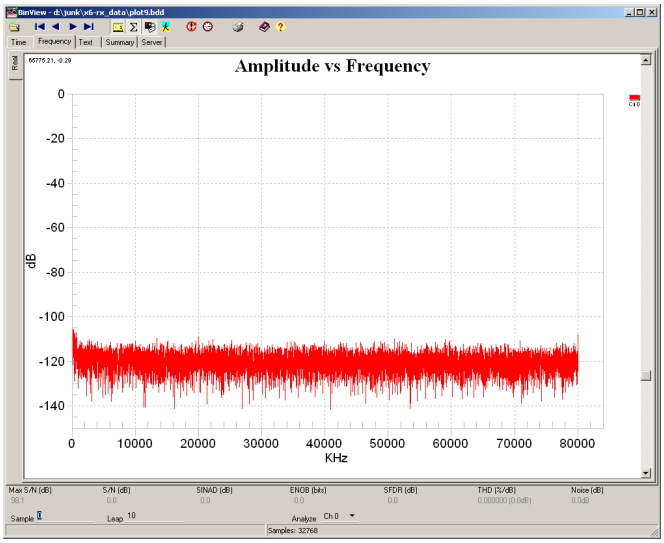
Signal quality, Fin = 70 MHz, Fs = 160 MHz onboard PLL. Channel 0, 22 pF parallel cap at A/D device inputs



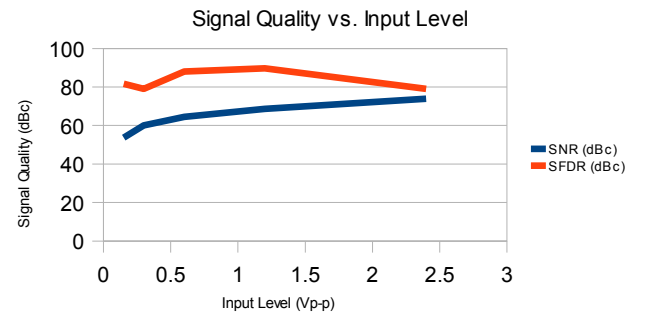
Signal quality, Fin = 70 MHz, Fs = 160 MHz onboard PLL. Channel 0, 22 pF parallel cap at A/D device inputs

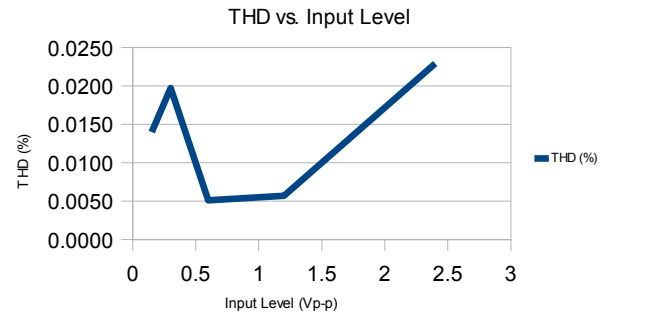


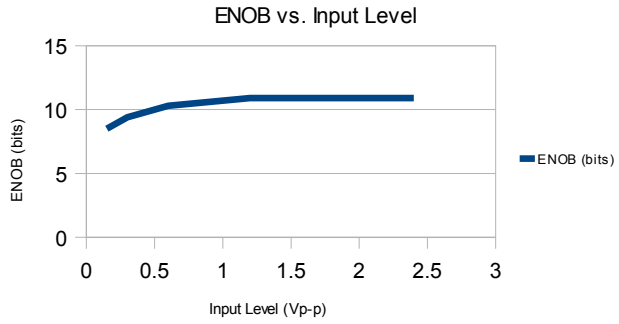
Signal quality, Fin = 70 MHz, Fs = 160 MHz onboard PLL. Channel 1, 70 MHz tank circuit at A/D device inputs, 160 MHz BPF at A/D device clock inputs



Signal quality, open input, Fs = 160 MHz external clock.







Connectors

Front Panel Connectors J1-J6

J1-J2connectors are positioned on the front panel for analog input, clock and trigger signals to be connected to the module.

Connector Type:	SMA 50 ohm
Number of Connections:	1 per signal
Connector Part Number	Amphenol 901-143
Mating Connector:	Amphenol 901-9511-3 or equivalent
Cable	Innovative part number 67048 SMA to BNC cable

Connector	Function
J1	A/D channel 0
J2	A/D channel 1
J3	A/D channel 2
J4	A/D channel 3

Connector	Function
J5	Trigger
J6	Sample / Reference Clock Input

Figure 43. Connectors J1-J6 Functions

XMC P15 Connector

P15 is the XMC PCI Express connector to the host.

Connector Types:	XMC pin header, 0.05 in pin spacing, vertical mount
Number of Connections:	114, arranged as 6 rows of 19 pins each
Connector Part Number	Samtec ASP-105885-01
Mating Connector:	Samtec ASP-105884-01

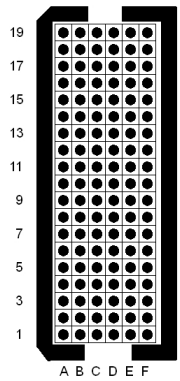


Figure 44. P15 XMC Connector Orientation

	Column					
Row	A	B	C	D	E	F
1	PET0p0	PET0n0	3.3V	PET0p1	PET0n1	VPWR
2	GND	GND		GND	GND	MRSTI#
3	PET0p2	PET0n2	3.3V	PET0p3	PET0n3	VPWR
4	GND	GND		GND	GND	MRSTO#
5	PET0p4	PET0n4	3.3V	PET0p5	PET0n5	VPWR
6	GND	GND		GND	GND	+12V
7	PET0p6	PET0n6	3.3V	PET0p7	PET0n7	VPWR
8	GND	GND		GND	GND	-12V
9						VPWR
10	GND	GND		GND	GND	GA0
11	PER0p0	PER0n0	MBIST#	PER0p1	PER0n1	VPWR
12	GND	GND	GA1	GND	GND	MPRESENT#
13	PER0p2	PER0n2	3.3VAUX	PER0p3	PER0n3	VPWR
14	GND	GND	GA2	GND	GND	MSDA
15	PER0p4	PER0n4		PER0p5	PER0n5	VPWR
16	GND	GND	MVMRO	GND	GND	MSCL
17	PER0p6	PER0n6		PER0p7	PER0n7	
18	GND	GND	FAN_TACH	GND	GND	3.3V
19	PEX REFCLK+	PEX REFCLK-	LED_N**	WAKE#	ROOT#	FAN#**

Table 62. X6-RX XMC Connector P15 Pinout

Note: All unlabeled pins are not used by X6 modules but may defined in VITA42 and VITA42.3 specifications.

**Note:LED_N, FAN#, and FAN_TACH are special purpose pins that support Innovative adapter card functions. These are reserved pins on the VITA42.3 specification.

Signal	Description
PET0px/PET0nx	PCI Express Tx +/-
PER0px/PER0nx	PCI Express Rx +/-
PEX REFCLK+/-	PCI Express reference clock, 100 MHz +/-
MRSTI#	Master Reset Input, active low
MRSTO#	Master Reset Output, active low
GA0	Geographic Address 0
GA1	Geographic Address 1
GA2	Geographic Address 2
MBIST#	Built-in Self Test, active low
MPRESENT#	Present, active low
MSDA	PCI Express Serial ROM data
MSCL	PCI Express Serial ROM clock
MVMRO	PCI Express Serial ROM write enable
WAKE#	Wake indicator to upstream device, active low
ROOT#	Root device, active low
LED_N#	Host LED control output, active low. (May be used with eInstruments)
FAN#	Host fan control. (May be used with eInstruments)
FAN_TACH	Host fan tachometer feedback. (May be used with eInstruments)

Table 63. P15 Signal Descriptions

XMC P16 Connector

P16 is the XMC secondary connector to the host and is used for digital IO, data link and triggering functions.

Connector Types:	XMC pin header, 0.05 in pin spacing, vertical mount
Number of Connections:	114, arranged as 6 rows of 19 pins each
Connector Part Number	Samtec ASP-105885-01
Mating Connector:	Samtec ASP-105884-01

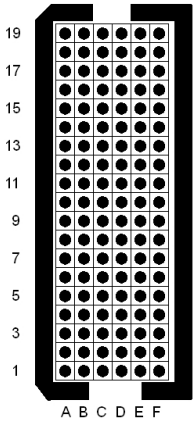


Figure 45. P16 XMC Connector Orientation

Table 64. X6-RX XMC Secondary Connector P16 Pinout

	Column					
Row	A	B	C	D	E	F
1	TXP0	TXN0	DIO_P0	TXP1	TXN1	DIO_P1
2	DGND	DGND	DIO_N0	DGND	DGND	DIO_N1
3	TXP2	TXN2	DIO_P2	TXP3	TXN3	DIO_P3
4	DGND	DGND	DIO_N2	DGND	DGND	DIO_N3
5	TXP4	TXN4	DIO_P4	TXP5	TXN5	DIO_P5
6	DGND	DGND	DIO_N4	DGND	DGND	DIO_N5
7	TCP6	TXN6	DIO_P6	TXP7	TXN7	DIO_P7
8	DGND	DGND	DIO_N6	DGND	DGND	DIO_N7
9	FMC_CLKP	FMC_CLKN	DIO_P8	FMC_REFCLKP	FMC_REFCLKN	DIO_P9
10	DGND	DGND	DIO_N8	DGND	DGND	DIO_N9
11	RXP0	RXN0	DIO_P10	RXP1	RXN1	DIO_P11
12	DGND	DGND	DIO_N10	DGND	DGND	DIO_N11
13	RXP2	RXN2	DIO_P12	RXP3	RXN3	DIO_P13
14	DGND	DGND	DIO_N12	DGND	DGND	DIO_N13
15	RXP4	RXN4	DIO_P14	RXP5	RXN5	DIO_P15
16	DGND	DGND	DIO_N14	DGND	DGND	DIO_N15
17	RXP6	RXN6	DIO_P16	RXP7	RXN7	DIO_P17
18	DGND	DGND	DIO_N16	DGND	DGND	DIO_N17
19	H_TRIG0	H_TRIG1	DIO_P18	H_PPS	FMC_VADJ	DIO_N18

Note: All unlabeled pins are not used by X6 modules but may defined in VITA42 and VITA42.3 specifications.

Table 65. P16 Signal Descriptions

Signal	Description
DIO_Px/Nx	Digital IO pairs +/-
TXP0-7	MGT transmit positive
TXN0-7	MGT transmit negative
RXP0-7	MGT receive positive
RXN0-7	MGT receive negative
DGND	Ground
FMC_CLKP/N	Sample clock input pair P/N; LVDS, 100 ohm differential termination.
FMC_REFCLKP/N	MGT reference clock input pair; LVDS, 100 ohm differential termination.
H_TRIG0-1	Trigger inputs.
H_PPS	PPS (pulse-per-second) input for trigger and timing controls.
F_ADJ	Reference voltage for DIO_Px/Nx inputs.

PMC JN1 Connector

JN1 implements part of the PMC PCI interface as well as power inputs.

Connector Types:	1mm double row, IEEE 1386 compatible vertical connector
Number of Connections:	64
Connector Part Number	Molex P/N 71436-2864
Mating Connector:	Molex P/N 71436

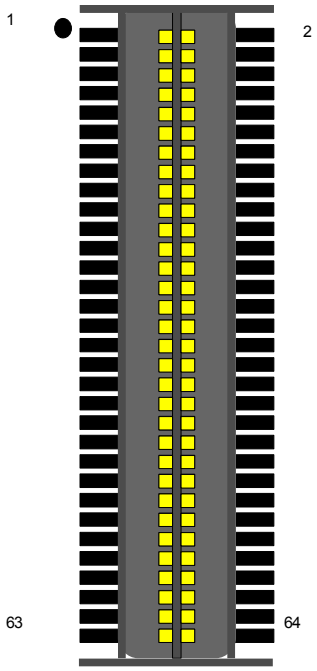


Figure 46. PMC JN1 Connector Schematic

See PCI specification for detailed pin descriptions.

Table 66. X6 JN1 Connector Pinout

Signal	Direction	Pin	Pin	Signal	Direction
-		1	2	-	-
DGND	P	3	4	INTA – interrupt to host	O
-		5	6	-	
-		7	8	+5V	P
-		9	10	-	-
DGND	P	11	12	-	-
CLK	I	13	14	-	-
DGND	P	15	16	GNT	P
REQ#	O	17	18	+5V	-
-		19	20	AD31	I/O
AD28	I/O	21	22	AD27	I/O
AD25	I/O	23	24	DGND	-
DGND		25	26	CBE3#	I/O
AD22	I/O	27	28	AD21	I/O
AD19	I/O	29	30	+5V	P
-		31	32	AD17	I/O
FRAME#	I/O	33	34	DGND	P
DGND	P	35	36	IRDY#	I/O
DEVSEL#	I/O	37	38	+5V	P
DGND	P	39	40	LOCK#	I/O
-		41	42	-	
PAR	I/O	43	44	DGND	P
-		45	46	AD15	I/O
AD12	I/O	47	48	AD11	I/O
AD9	I/O	49	50	+5V	P
DGND	P	51	52	CBE0#	I/O
AD6	I/O	53	54	AD5	I/O
AD4	I/O	55	56	DGND	I/O
-		57	58	AD3	I/O
AD2	I/O	59	60	AD1	I/O
AD0	I/O	61	62	+5V	P

Signal	Direction	Pin	Pin	Signal	Direction
DGND		63	64	-	-

PMC JN2 Connector

JN2 implements part of the PMC PCI interface as well as power inputs.

Connector Types:	1mm double row, IEEE 1386 compatible vertical connector
Number of Connections:	64
Connector Part Number	Molex P/N 71436-2864
Mating Connector:	Molex P/N 71436

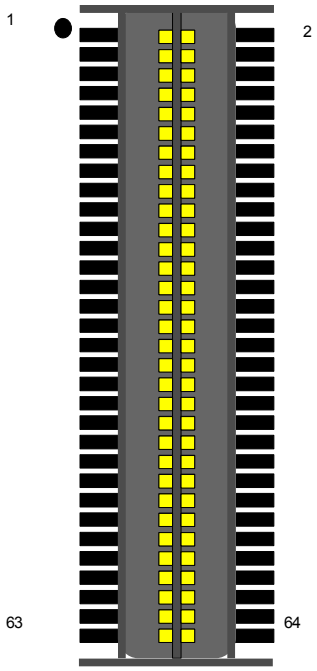


Figure 47. PMC JN2 Connector Schematic

See PCI specification for detailed pin descriptions.

Table 67. X6 JN2 Connector Pinout

Signal	Direction	Pin	Pin	Signal	Direction
+12V	P	1	2	-	
-		3	4	-	
-		5	6	DGND	P
DGND	P	7	8	-	
-		9	10	-	
-		11	12	3.3V	P
RST#	I	13	14	-	
3.3V	P	15	16	-	
-		17	18	DGND	P
AD30	I/O	19	20	AD29	I/O
AD24	I/O	21	22	AD26	I/O
IDSEL	I	23	24	3.3V	P
DGND	P	25	26	AD23	I/O
3.3V	P	27	28	AD20	I/O
AD118	I/O	29	30	DGND	P
AD16	I/O	31	32	CBE2#	I/O
DGND	P	33	34	-	
TRDY#	I/O	35	36	3.3V	P
DGND	P	37	38	STOP	I/O
PERR#	O	39	40	DGND	P
3.3V	P	41	42	SERR#	O
CBE1#	I/O	43	44	DGND	P
AD14	I/O	45	46	AD13	I/O
M66EN	O	47	48	AD10	I/O
AD8	I/O	49	50	3.3V	P
AD7	I/O	51	52	-	
3.3V	P	53	54	-	
-		55	56	DGND	P
-		57	58	-	
DGND	P	59	60	-	
-		61	62	3.3V	P

Signal	Direction	Pin	Pin	Signal	Direction
DGND	P	63	64	-	-

PMC JN4 Connector

JN4 implements digital IO connections.

Connector Types:	1mm double row, IEEE 1386 compatible vertical connector
Number of Connections:	64
Connector Part Number	Molex P/N 71436-2864
Mating Connector:	Molex P/N 71436

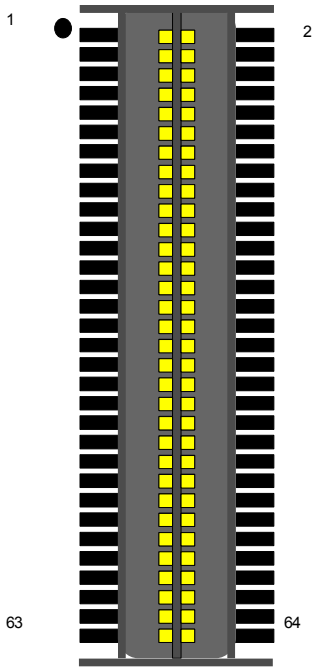


Figure 48. PMC JN4 Connector Schematic

Table 68. X6 JN2 Connector Pinout

Signal	Direction	Pin	Pin	Signal	Direction
DIO_P0	I/O	1	2	DIO_P1	I/O
DIO_N0	I/O	3	4	DIO_N1	I/O
DIO_P2	I/O	5	6	DIO_P3	I/O
DIO_N2	I/O	7	8	DIO_N3	I/O
DIO_P4	I/O	9	10	DIO_P5	I/O
DIO_N4	I/O	11	12	DIO_N5	I/O
DIO_P6	I/O	13	14	DIO_P7	I/O
DIO_N6	I/O	15	16	DIO_N7	I/O
DIO_P8	I/O	17	18	DIO_P9	I/O
DIO_N8	I/O	19	20	DIO_N9	I/O
DIO_P10	I/O	21	22	DIO_P11	I/O
DIO_N10	I/O	23	24	DIO_N11	I/O
DIO_P12	I/O	25	26	DIO_P13	I/O
DIO_N12	I/O	27	28	DIO_N13	I/O
DIO_P14	I/O	29	30	DIO_P15	I/O
DIO_N14	I/O	31	32	DIO_N15	I/O
DIO_P16	I/O	33	34	DIO_P17	I/O
DIO_N16	I/O	35	36	DIO_N17	I/O
DIO_P18	I/O	37	38	DIO_P19	I/O
DIO_N18	I/O	39	40	DIO_N19	I/O
DIO_P20	I/O	41	42	DIO_P21	I/O
DIO_N20	I/O	43	44	DIO_N21	I/O
DIO_P22	I/O	45	46	DIO_P23	I/O
DIO_N22	I/O	47	48	DIO_N23	I/O
DIO_P24	I/O	49	50	DIO_P25	I/O
DIO_N24	I/O	51	52	DIO_N25	I/O
DIO_P26	I/O	53	54	DIO_P27	I/O
DIO_N26	I/O	55	56	DIO_N27	I/O
DIO_P28	I/O	57	58	DIO_P29	I/O
DIO_N28	I/O	59	60	DIO_N29	I/O
DIO_P30	I/O	61	62	DIO_P31	I/O

Signal	Direction	Pin	Pin	Signal	Direction
DIO_N30	I/O	63	64	DIO_N31	I/O

Xilinx JTAG Connector

JP3 is used for the Xilinx JTAG chain. It connects directly with Xilinx JTAG cables such as Parallel Cable IV or Platform USB.

Connector Types:	14-pin dual row male header, 2mm pin spacing, right angle
Number of Connections:	14, arranged as 2 rows of 7 pins each
Connector Part Number	Samtec TMM-107-01-L-D-RA or equivalent
Mating Connector:	AMP 111623-3 or equivalent

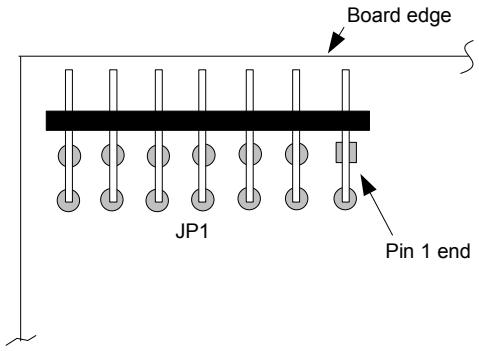


Figure 49. X6-RX JP3 Orientation, board face view

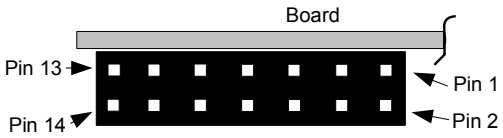


Figure 50. X6-RX JP3 Orientation, board top edge view

Pin	Signal	Direction
1,3,5,7,9,11,13	Digital Ground	Power
2	1.8V	Power
4	TMS	I
6	TCK	I
8	TDO	O
10	TDI	I
12,14	No Connect	-

Table 69. X6-RX JP3 Xilinx JTAG Connector Pinout

FLASH Boot Image Select

JP1 is used to select the logic image used to boot the module.

JP3 Strapping	Boot Image
None	Application Logic Image
1-2	Backup Image (Golden Image)

Table 70. X6-RX JP1 Boot Image Select

Mechanicals

The following diagrams show the X6-RX connectors and physical locations. The XMC conforms to IEEE 1386 form factor, 75mm x 150mm. The spacing to the host card is 10 mm and consumes a single slot in desktop and Compact PCI/PXI chassis. An EMI shield over the analog section is normally installed.

Detailed drawings for mechanical design work are available through technical support.

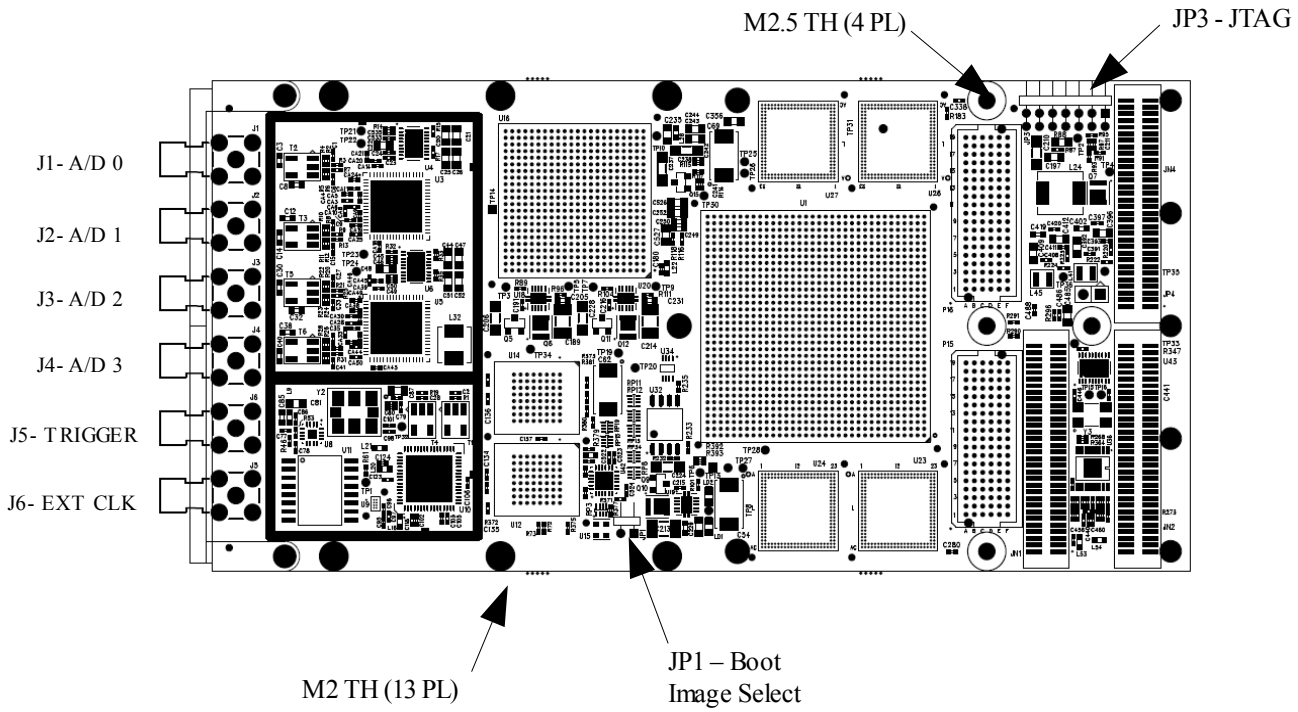

M2.5 TH (4 PL)
JP3 - JTAG
J1- A/D 0
J2- A/D 1
J3- A/D 2
J4- A/D 3
J5- TRIGGER
J6- EXT CLK
M2 TH (13 PL)
JP1 – Boot
Image Select

Figure 51. X6-RX Mechanicals (Top View)

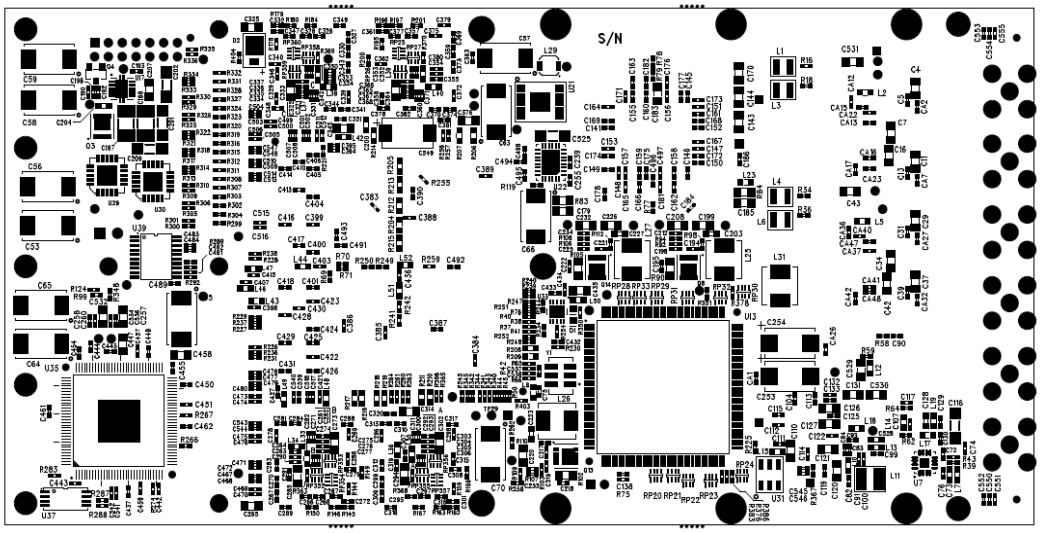


Figure 52. X6-RX Mechanicals (Bottom View)